IBM Text-to-Speech

API Reference

Version 6.4.0

Printed in the USA

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

Twelfth Edition (March 2002)

Contents

About This Book

This book provides information on incorporating IBM Text-to-Speech technology into other applications. It describes the programming interfaces available for developers to take advantage of these features within their applications. This book is prepared in Portable Document Format (PDF) to provide the advantages of text search and cross-reference hyperlinking and is viewable with the Adobe Acrobat Reader v.3.x or higher. We recommend that you print all or part of this guide for quick reference.

Who Should Read This Book?

Read this book if you are a software developer interested in writing applications that use IBM Text-to-Speech technology. This document describes the use of IBM Text-to-Speech technology for beginning to advanced software engineers.

Organization of This Book

This document is organized in the following manner:

- **"The IBM Text-to-Speech Software Developer's Kit"** contains general information about the structure and organization of the IBM Text-to-Speech SDK, including an overview of the API interfaces and a description of the SDK-provided tools.
- **"The ECI Application Programming Interface"** contains information about using IBM Text-to-Speech with its proprietary "Eloquence Command Interface" API.
- **"ECI Reference"** contains detailed information about the data types and functions available for use with the Eloquence Command Interface.
- **"Annotations"** includes a description of the use of special codes that can be inserted into the input text to customize the behavior of IBM Text-to-Speech .
- **"Symbolic Phonetic Representations"** describes the use of special phonetic symbols to customize pronunciations in IBM Text-to-Speech.

- **“Glossary of Linguistic Terms”** contains definitions of linguistic terms used in this manual.

Typographical Conventions

The following typographical conventions are used throughout this document to facilitate reading and comprehension. They are outlined in the following table.

Text Format	Applies to
`Monospace font`	Code samples, file and directory names.
Bold	Function and callback names; data types (including structures and enumerations).
Italics	Parameter and structure member names; sample text; the introduction of a new term.
`UPPERCASE`	Property, enumerator, mode, and state names.

The IBM Text-to-Speech Software Developer's Kit

Overview

The IBM TTS allows you to incorporate high-quality text-to-speech functionality into your applications. This SDK offers developers the application programming interfaces (APIs) for the proprietary, platform-independent Eloquence Command Interface (ECI). The typical installation of the IBM Text-to-Speech SDK, which includes this document, along with the IBM TTS RunTime, provides all the necessary software and support files for these APIs.

The following sections include a brief description of each of the available APIs and directory structure of this SDK.

Eloquence Command Interface (ECI)

The Eloquence Command Interface (ECI) is a proprietary, platform independent API that allows direct access to all the functionality and power of the IBM Text-to-Speech. This API:

- Is supported on a variety of operating systems.
- Allows customization of speech output both through function calls and textual annotations.
- Does not use the Windows Registry to find components, allowing developers to include a private copy of the text-to-speech engine with their application that is less likely to be accidentally modified by later installations or by other applications.

See the sections The ECI Application Programming Interface and ECI Reference for details on how to use this API. See the section Annotations for details on the use of ECI annotations to customize speech output.

The ECI Application Programming Interface

Overview

The Eloquence Command Interface (ECI) is a library that provides an interface between applications and the IBM Text-to-Speech system. Version 6.2 of ECI has been re-architected to provide support for multiple concurrent speech synthesis threads, and a consistent interface on all supported platforms.

As in prior versions of ECI, text is appended to the input buffer. Each word takes its voice definition from the active voice. Speech is synthesized from the input buffer according to the associated voice parameters, placed in the output audio buffer, and sent to the appropriate destination. The active voice can be set from a number of built-in voices or from a user-defined voice. Language, dialect, and voice parameters can be modified individually using either ECI function calls or annotations inserted into the input buffer with the input text. As text is added to the input buffer, the active voice definition is stored with it, so that changes to the active voice do not affect text already in the input buffer.

Indices can be used to determine when the delimited text fragment has been synthesized. A message will be received when all text inserted before the index has been synthesized.

Output can be sent to one of three types of destinations: a callback function, a file, or an audio device. These destination types are mutually exclusive, so sending output to one of them turns off output to the previous destination. The default destination is an available audio device.

Structuring an ECI Program

Using eciSpeakText for Simple Programs

The simplest way to incorporate text-to-speech into your application is by using the high-level ECI function **eciSpeakText**, which speaks the given text to the default audio device. This first sample C program speaks a short phrase and then exits:

```
#include <eci.h>

int main(int argc, char *argv[])
  {
    eciSpeakText ("Hello World!", false);

    return 0;
  }
```

Managing an ECI Instance

In order to use the more powerful features of the ECI API, you will have to manage ECI instances directly. An ECI instance, in accordance with standard object-oriented procedure, originates with a call to **eciNew** and ends with a call to **eciDelete**.

One basic strategy for managing an ECI instance is outlined below:

- Create a new ECI instance by invoking **eciNew**.
- If you want ECI to notify you of certain events, register a callback function with a call to **eciRegisterCallback**.
- Interact with the ECI instance. You may, for example:
  - Add text to the ECI instance's input buffer with one or more calls to **eciAddText**.
  - To synthesize annotated text, call **eciSetParam(*eciInstance*, eciInputType,** 1**)** before calling **eciAddText.** This lets ECI know that the text may contain annotations.
  - To use one of the preset voices, call **eciCopyVoice** before calling **eciAddText.** The active voice (voice 0) specifies values for a set of voice characteristics, such as pitch baseline and

pitch fluctuation, which are applied to all new text added to the input buffer. See Voice Parameters for more detailed discussion.

- Change the state of the active voice with calls to **eciSetVoiceParam**.

- Call **eciSynthesize** when all text has been added to the input buffer. To synthesize text in line-oriented format, such as a table or list, call **eciAddText** and **eciSynthesize** for each line, to ensure that each line is spoken as a separate sentence.
- If the thread that is managing this ECI instance does not contain a Windows message loop, you must ask your instance of ECI to report that synthesis is complete. This step will also allow your registered callback to be called by ECI. You can do this in more than one way:
  - Call **eciSynchronize**, which waits in an efficient state, allowing callbacks to be called, until synthesis is finished. When synthesis is complete, the function will return control to the calling thread. Do not call **eciSynchronize** from a thread that has a Windows message loop.
  - Call **eciSpeaking** until it returns `false`. Each call to **eciSpeaking** will allow your callback to be called.

  If the thread that is managing this ECI instance contains a Windows message loop, this step is not necessary.
- Use **eciDelete** to free the resources dedicated to your instance.

The following example speaks a phrase in English, then a phrase in French, then exits:

```
#include <eci.h>

int main(int argc, char *argv[])
{

  ECIHand eciHandle;

  eciHandle = eciNew();//Create a new ECI Instance
  if (eciHandle!= NULL_ECI_HAND) //Success?
    {//Give some text to the instance
    if (!eciAddText(eciHandle, "Hello World!"))
    {
      //We failed to add text
      //Print an error message
      printf( "eciAddText failed\n" );
    }
//Change the language to Standard French,
//if available
if (eciSetParam(eciHandle, eciLanguageDialect,
eciStandardFrench) == -1)
{
//Error Changing to French
printf( "Could not change to French\n" );
}
else
{
//Give some text in French
if (!eciAddText(eciHandle, "Un. Deux. Trois."))
{
//We failed to add text
//Print an error message
printf( "eciAddText failed\n" );
}


Continued on the next page
```

```
   Continued from previous page
//Start ECI speaking
if (!eciSynthesize(eciHandle))
{
 //We failed to synthesize
 //Print an error message
printf( "eciSynthesize failed\n" );
}
}
//Wait until ECI finishes speaking
if (!eciSynchronize(eciHandle))
{
//We failed to synchronize
//Print an error message
printf( "eciSynchronize failed\n" );
}
//Delete our ECI Instance; deallocates memory
eciDelete(eciHandle);
}
else
{
//We failed to create a new ECI Instance
//Print an error message
printf( "eciNew failed\n" );
}

return 0;
}
```

Threading

The ECI API is structured on a principle called the "Single-Threaded Apartment Model", which means that each individual instance can be called only upon the thread that created it; that is, it should not be affected by the existence of other instances or threads. All callbacks are called by the thread that created the instance.

The **eciSpeakText** function is a *blocking* function that creates, manages, and destroys its own private ECI instance. The application thread of execution is blocked until the function returns. **eciSpeakText** requires no special thread handling, since it does not return control to the main thread until it has completed all synthesis.

Other ECI functions are *non-blocking*: the application thread of execution remains available during their execution. Applications using animated mouths, multiple voices, multiple conversations or requiring the highest possible performance depend on these non-blocking functions, which are only accessible through the handle created by **eciNew**. See also **eciNewEx**.

Callbacks

A *callback* is a mechanism for temporarily passing control of execution out of an instance of ECI to a function provided by the developer when certain events take place. The ECI API provides for four callback events:

- *eciIndexReply*: Sends notification when a particular point in the input text is reached. To set these points in the text, call **eciInsertIndex** after calls to **eciAddText**.
- *eciPhonemeBuffer*: Sends notification when the Symbolic Phonetic Representations buffer is full. Call **eciGeneratePhonemes** after a call to **eciAddText** to enable this event.
- *eciPhonemeIndexReply*: Sends notification when a particular phoneme is spoken, including mouth animation data for that phoneme. Set **eciWantPhonemeIndices** to 1 with **eciSetParam** to enable this event.
- *eciWaveformBuffer*: Sends notification when a sample-capture buffer is full (so, e.g., the developer can send the samples to a custom audio destination). Call **eciSetOutputBuffer** to enable this event.

Only one callback function may be registered for each instance of ECI. This function will receive all four types of callback events. No events are set by default.

Callback functions must return promptly, returning a flag indicating completion of processing. Callbacks may not call ECI functions.

Register your callback with **eciRegisterCallback** immediately after calling **eciNew**. For any given ECI instance, your callback will be called from the same thread on which your application calls ECI. See eciRegisterCallback for more details on use of callbacks.

User Dictionaries

IBM TTS allows you to explicitly specify pronunciations for words, abbreviations, acronyms, and other sequences, preventing the normal pronunciation rules from applying. One way you can do this is to enter a Symbolic Phonetic Representation (SPR) annotation directly into the input text (see Symbolic Phonetic Representations). A more permanent way is to enter the word (the input string or *key*) and the pronunciation you want (the output or *translation value*) in one of the user dictionaries.

A *dictionary set* consists of 4 *volumes*. Each volume differs from the kinds of keys and translation values it accepts.

- Main Dictionary (eciMainDict)
- Main Extension Dictionary (eciMainDictExt)
- Roots Dictionary (eciRootDict)
- Abbreviations Dictionary (eciAbbvDict)

A dictionary file consists of ASCII text with one dictionary entry per line. Each input line contains a key and a translation value, separated by a tab character. An invalid key or translation will cause the dictionary look-up to fail, and the pronunciation of the word will be generated by the normal pronunciation rules. Valid entries for each dictionary are discussed in the subsections below.

To add, modify, or delete an entry in any of the dictionaries, use the **eciUpdateDict** function of the API.

For Asian languages, such as Chinese and Japanese, the client application should use the dictionary maintenance functions that are named with an A at the end in place of the same-name function. For example, use **eciDictFindFirstA**, in stead of **eciDictFindFirst**.

ForChinese, Roots Dictionary (eciRootDict) functionality is not supported.

Main Dictionary (eciMainDict)

The Main Dictionary is distinguished from the other user dictionaries in two ways: a valid translation consists of any valid input string, and the key of a Main Dictionary entry may contain any characters other than white space, except that the final character of the key may not be a punctuation symbol. You can thus use the Main Dictionary for:

- Strings that translate into more than one word
- Keys that require translations which include annotations or SPRs
- URLS and email addresses
- Keys containing digits or other non-letter symbols
- Acronyms with special pronunciations

The Main Dictionary is case-sensitive. For example, if you enter the key "WHO" with the translation "World Health Organization", lower case who will still be pronounced as expected (`[hu]).

Note: The Main Dictionary translations may include ECI annotations.

Valid Main Dictionary Entries

The following table summarizes the valid Main Dictionary keys and translations:

Key	Translation
· letters, both upper and lower case · digits · non-alphanumeric characters like @, #, $, %, &, *, + · apostrophes, quotation marks, parentheses, brackets, etc. · punctuation, except as the final character	Anything that is legal input to the text-to-speech engine, including white space, punctuation, SPRs, and annotations.
NO: white space	

Main Dictionary Examples

The following table shows examples of Main Dictionary entries:

Key	Translation
AWSA	American Woman Suffrage `0 Association
jeb@notreal.org	j e b at not real dot o r g
ECSU	`[1i] `[1si] `[1Es] `[1yu]
UConn	`[2yu1kan]
WYSIWYG	`[1wI0zi0wIg]
Win32	win thirty two
486DX	4 86 dee ecks

See Also

Abbreviations Dictionary (eciAbbvDict), Roots Dictionary (eciRootDict).

Main Extension Dictionary (eciMainDictExt)

The Main Extension Dictionary is the used for Asian languages and provides support for Chinese, Japanese, and Korean.

You can use the Main Extension Dictionary for:

- Strings for DBCS languages (other than white space)
- Strings that translate into more than one word
- Keys that require translations which include annotations or SPRs
- Keys containing digits or other non-letter symbols
- Acronyms with special pronunciation

Translation is language dependent. For example in Japanese, Katakana Yomi strings are valid translations. Any other SBCS/DBCS characters except the accent mark (^) will cause an error.

Each Main Extension Dictionary entry requires a part of speech which specifies the grammatical category. The possible values are:

Language	Part of Speech (POS)
Chinese	eciUndefinedPOS eciMingCi
Japanese	eciUndefinedPOS eciFutsuuMeishi eciKoyuuMeishi eciSahenMeishi
Korean	eciUndefinedPOS

Note: The Main Extension Dictionary can be accessed with eciUpdateDictA, eciDictFindFirstA, eciDictFindNextA, eciDictLookupA.

Roots Dictionary (eciRootDict)

The Roots Dictionary is used for ordinary words, like nouns (including proper names), verbs, or adjectives, and for proper names. The distinctive feature of the Roots Rictionary is that you only have to enter the root form of a word; all other forms of the word will automatically get pronounced in the same way. For example, the letter-to-sound rules normally pronounce *roof* as [ruf] (which has the vowel of *boot*). You can use the Roots Dictionary to specify the alternate pronunciation [rUf] (which has the vowel of *book*). Then, all words with this root, such as *roofer* and *roofing* will also be pronounced this way; there is no need to list the other words separately in the dictionary.

- The Roots Dictionary is not case-sensitive. So, for example, when you enter a root in lowercase, it will still be found and pronounced as specified even when it begins with an uppercase (capital) letter (for example, as the first word in a sentence).
- The Roots Dictionary is designed to provide alternate pronunciations of *existing roots*, and may not work properly in the case of unknown roots. For example, the entry *prego* occurring in the hypothetical word *pregoness* will not be accessed from the user roots dictionary because the linguistic analysis rules assume that the word contains the root *go* rather than the root *prego*.
- The roots dictionary cannot be used to specify an alternate pronunciation of a function word, such as *the* or *to*.

Valid Roots Dictionary Entries

The following table summarizes valid Roots Dictionary keys and translations:

Keys	Translations
A single word in ordinary spelling, all lowercase letters	· A single word in ordinary spelling · A valid SPR
NO: digits, punctuation, white space, or other non-letter characters	**NO**: digits, punctuation, or other non-letter characters, white space, tags, or annotations

Roots Dictionary Examples

The following table shows examples of Roots Dictionary entries:

Key	Translation	Would apply to:
figure	`[.1fI.0gR]	figures, figuring, figured, refigure
tomato	`[.0tx.1ma.0to]	tomatoes, tomato’s

Key	Translation	Would apply to:
wash	`[.1warS]	wash, washing, washed, washes
wilhelmina	wilma	Wilhelmina, Wilhelmina’s

See Also

Main Dictionary (eciMainDict), Abbreviations Dictionary (eciAbbvDict)

Abbreviations Dictionary (eciAbbvDict)

The Abbreviations Dictionary is used for abbreviations (both with and without periods) which do not require the use of annotations in their translation.

The Abbreviations Dictionary is case-sensitive. So for example, if you entered the key Mar with translation "march," lower-case "mar" would still be pronounced as expected (`[mar]).

When you enter a key in the Abbreviations Dictionary, it is not necessary to include the "trailing" period (as in the final period of "etc."). However, if you want an abbreviation to be pronounced as specified in the translation *only* when it is followed by a period in the text, then you must enter the trailing period in the key. The following table summarizes the use of trailing periods:

Key entry:	Will match:
`inv`	inv. inv
`sid.`	sid. (not sid)

An Abbreviations Dictionary entry invokes different assumptions about how to interpret the trailing period in the text than does a Main Dictionary entry. Since the period cannot be part of a Main Dictionary entry key, it is automatically interpreted as end-of-sentence punctuation. A period following an Abbreviations Dictionary entry, on the other hand, is ambiguous. It will only be interpreted as end-of-sentence punctuation if other appropriate conditions obtain (e.g., if it is followed by two spaces and an upper-case letter). For example, input (a) will be interpreted as one sentence, while (b) will be interpreted as two sentences.

```
(a) It rained 2 cm. on Monday.
(b) On Sunday it rained 2 cm.  On Monday, it was sunny.
```

Valid Abbreviations Dictionary Entries

The following table summarizes valid Abbreviations Dictionary keys and translations:

Keys	Translation
• Sequences of one or more letters separated by periods (x.x.x. or xx.xx.xx) • Sequences of letters, with or without the trailing period that may be considered part of the abbreviation (xxx. or xxx) • Upper or lower case letters • Internal apostrophes (not the first or last character in the sequence)	• One or more valid words in ordinary spelling, including both upper and lower case letters, separated by white space or hyphen
NO: digits, non-letter symbols, white space, or punctuation other than periods	**NO:** digits, punctuation, SPRs, tags, or annotations

Abbreviations Dictionary Examples

The following table shows examples of Abbreviations Dictionary entries:

Key	Translation
Is.D.	eye ess dee
punct	punctuation
para	paragraph
Ltjg	lieutenant junior-grade
Fr	Friar
int'l	international

See Also

Main Dictionary (eciMainDict), Roots Dictionary (eciRootDict).

You can temporarily override the use of both internal and user-defined abbreviations with an annotation; see Dictionary Processing of Abbreviations.

ECI Reference

This section contains the following reference information:

Data Types

ECI defines the following data types in the header file `eci.h` which should be included in any source file that uses ECI functions.

Boolean

```
typedef int Boolean;
```

Many ECI functions return Boolean values.

ECICallbackReturn

```
typedef enum{
  eciDataNotProcessed,
  eciDataProcessed
  eciDataAbort
}ECICallbackReturn
```

If you register a callback function, it must return one of these enumerated values.

ECIDictError

```
typedef enum{
```

DictNoError,
The call executed properly.

DictNoEntry,
The dictionary is empty, or there are no more entries.

DictFileNotFound,
The specified file could not be found.

DictOutOfMemory,
Ran out of heap space when creating internal data structures.

DictInternalError,
An error occurred in the internal synthesis engine.

DictAccessError
An error occurred when claiming operating-system specific resources for dictionary access.

DictErrLookUpKey
An error occurred when looking up the key.

DictInvalidVolume
The dictionary volume is not supported by the current language.

```
}ECIDictError
```

Most dictionary volume access functions return a value of this type to report errors.

ECIDictHand

```
typedef void* ECIDictHand
```

A handle to an ECI dictionary set.

ECIDictVolume

```
typedef enum {
   eciMainDict,
   eciRootDict,
   eciAbbvDict,
   eciMaindDictExt
}ECIDictVolume;
```

Identifies dictionary set volumes. See User Dictionaries.

ECIFilterError

```
typedef enum {
  FilterNoError,
```
The call executed properly.

```
  FilterFileNotFound,
```
The specified filter could not be found.

```
  FilterOutOfMemory,
```
Ran out of heap space when creating internal data structures

```
  FilterInternalError,
```
An error occurred in the internal synthesis engine.

```
  FilterAccessError,
```
An error occurred when claiming operating-system specific resources for filter access.

```
} ECIFilterError
```

ECIHand

```
typedef void* ECIHand
```

A handle to an instance of ECI.

ECIInputText

```
typedef const void* ECIInputText
```

Contains an NULL terminated string using a system-dependent character set (currently ANSI for all platforms).

ECILanguageDialect

```
typedef enum {
  eciGeneralAmericanEnglish,
  eciBritishEnglish,
  eciCastilianSpanish,
  eciMexicanSpanish,
  eciStandardFrench,
  eciCanadianFrench
  eciStandardGerman,
  eciStandardItalian,
  eciMandarinChinese,
  eciTaiwaneseMandarin,
  eciBrazilianPortuguese
  eciStandardJapanese,
  eciStandardFinnish,
  eciStandardNorwegian
  eciStandardSwedish,
  eciStandardDanish
} ECILanguageDialect
```

Identifies a language and dialect.

ECIMessage

```
typedef enum{
  eciWaveformBuffer,
  eciPhonemeBuffer,
  eciIndexReply,
  eciPhonemeIndexReply
}ECIMessage
```

Indicates why a callback has been called.

ECIParam

```
typedef enum{
   eciSynthMode,
   eciInputType,
   eciTextMode,
   eciDictionary,
   eciSampleRate,
   eciWantPhonemeIndices,
   eciRealWorldUnits,
   eciLanguageDialect,
   eciNumberMode,
   eciPhrasePrediction,
   eciNumParams
}ECIParam
```

Specifies a synthesis state parameter for function calls that get and set synthesis state attributes.

ECIVoiceParam

```
typedef enum{
  eciGender,
  eciHeadSize,
  eciPitchBaseline,
  eciPitchFluctuation,
  eciRoughness,
```

```
  eciBreathiness,
  eciSpeed,
  eciVolume,
  eciNumVoiceParams
}ECIVoiceParam
```

Specifies a voice parameter for function calls that get and set voice attributes.

ECIMouthData

Consists of a phoneme, language and dialect of the phoneme, and mouth position data for the phoneme. Returned by callbacks with the **eciPhonemeIndexReply** message. See **eciRegisterCallback** for more details.

In addition to the phoneme symbols defined for SPR input, the symbol ¤ (0xA4) is also used to indicate end of utterance, and is sent with a set of neutral mouth position parameters.

```
typedef struct {
  char szPhoneme[eciPhonemeLength+1];
  ECILanguageDialect eciLanguageDialect;
  unsigned char mouthHeight;
  unsigned char mouthWidth;
  unsigned char mouthUpturn;
  unsigned char jawOpen;
  unsigned char teethUpperVisible;
  unsigned char teethLowerVisible;
  unsigned char tonguePosn;
  unsigned char lipTension;
} ECIMouthData;
```

Members

szPhoneme
: Null-terminated, ASCIIZ string containing the name of a phoneme, or ¤ (0xA4) for end-of-utterance.

eciLanguageDialect
: Language and dialect of this phoneme.

mouthHeight

Height of the mouth and lips. This is a linear range from 0-255, where 0 = minimum height (that is, mouth and lips are closed) and 255 = maximum possible height for the mouth.

mouthWidth
Width of the mouth and lips. This is a linear range from 0-255, where 0 = minimum width (that is, the mouth and lips are puckered) and 255 = maximum possible width for the mouth.

mouthUpturn
Extent to which the mouth turns up at the corners, that is, how much it smiles. This is a linear range from 0-255, where 0 = mouth corners turning down, 128 = neutral, and 255 = mouth is fully upturned.

jawOpen
Angle to which the jaw is open. This is a linear range from 0-255, where 0 = fully closed, and 255 = completely open.

teethUpperVisible
Extent to which the upper teeth are visible. This is a linear range from 0-255, where 0 = upper teeth are completely hidden, 128 = only the teeth are visible, and 255 = upper teeth and gums are completely exposed.

teethLowerVisible
Extent to which the lower teeth are visible. This is a linear range from 0-255, where 0 = lower teeth are completely hidden, 128 = only the teeth are visible, and 255 = lower teeth & gums are completely exposed.

tonguePosn
Tongue position. This is a linear range from 0-255, where 0 = tongue is completely relaxed, and 255 = tongue is against the upper teeth.

lipTension
Lip tension. This is a linear range from 0-255, where 0 = lips are completely relaxed, and 255 = lips are very tense.

Remarks

The inventory of phoneme symbols used as the values of szPhoneme is similar but not necessarily identical to the inventory of Symbolic Phonetic Representations (SPR) phoneme symbols. The values of szPhoneme are taken directly from the phonemic representation generated by the IBM Text-to-Speech TTS engine, whereas the symbols used in SPRs are normalized versions of these phonemes.

In addition to the phoneme symbols used in each language, the symbol ¤ (0xA4) is used to indicate the end of a sentence and is sent with a set of neutral mouth position parameters.

Synthesis State Parameters

When you create a new ECI instance, it is given a default synthesis state. As you interact with the instance, its state changes. You can:

- Get the current synthesis state using **eciGetParam**.
- Set the synthesis state directly, through **eciSetParam**, or indirectly, by sending annotated text in calls to **eciAddText**.

This section describes the synthesis state parameters that can be passed to **eciGetParam** and **eciSetParam**.

eciDictionary

0: Abbreviations dictionaries (both internal and user) are used (default).

1: Abbreviations dictionaries (both internal and user) are not used.

Enables or disables the internal and user abbreviations dictionaries. You can also turn abbreviations dictionary lookups on and off the using the ‘daN annotation (see Dictionary Processing of Abbreviations).

eciInputType

0: Plain: input consists of unannotated text. Any annotations will be spelled out (e.g., `v2 will be pronounced "backquote vee two") (default).

1: Annotated: input text includes annotations. See Annotations for more details.

eciLanguageDialect

```
enum
  {
  eciGeneralAmericanEnglish,
```

```
  eciBritishEnglish,
  eciCastilianSpanish,
  eciMexicanSpanish,
  eciStandardFrench,
  eciStandardGerman,
  eciStandardItalian,
  eciMandarinChinese,
  eciTaiwaneseMandarin,
  eciBrazilianPortuguese
  eciStandardJapanese,
  eciStandardFinnish,
  eciStandardKorean
} ECILanguageDialect
```

A value specifying the language and dialect. These should be of type **ECILanguageDialect**. Not all languages are available with all installations. The language defaults to the “lowest-numbered” language installed on the system. Languages are numbered in the order specified by the **ECILanguageDialect** enum.

This parameter can be set by the ‘lN annotation; see Selecting a Language and Dialect for more detail.

eciNumberMode

0: Pronounce 4-digit numbers as “nonyears” (e.g., “1984” would be pronounced “one thousand nine hundred eighty four”).

1: Pronounce 4-digit numbers as “years” (e.g., “1984” would be pronounced “nineteen eighty four”) (default)

This parameter can be set by the ‘tyN annotation; see Specifying Alternative Pronunciations for more detail.

eciNumParams

Total number of ECIParams. Passing eciNumParams to eciGetParam will cause a -1 (error) return, which is an expected behavior.

eciRealWorldUnits

0: Use ECI values (default).

1: Use Real World units.

Selects the units for the values of the voice parameters **eciPitchBaseline**, **eciSpeed**, and **eciVolume** as either ECI units or Real World units.

eciSampleRate

0: 8000 samples per second.

1: 11,025 samples per second (default).

2: 22,050 samples per second.

eciSynthMode

0: Sentence: The input buffer is synthesized and cleared at the end of each sentence (default).

1: Manual: Synthesis and input clearing is controlled by commands only.

eciTextMode

0: Default: no special interpretation (default).

1: AlphaSpell: letters and digits are spelled out, punctuation is treated normally to identify ends of phrases and sentences, and other symbols are ignored.

2: AllSpell: all symbols are spelled out. Note that sentence ends are not recognized in this mode.

3: IRCSpell: like AlphaSpell, except that letters are spelled out using the International Radio Code (“alpha, bravo, charlie”) rather than their conventional names.

This corresponds to the annotation ‘tsN, described in Specifying Alternative Pronunciations.

eciWantPhonemeIndices

0: Phoneme indices are not generated. (default)

1: If a callback has been registered (see eciRegisterCallback below), phoneme indices will be sent to the callback as each phoneme is being spoken. See also the **eciPhonemeIndexReply** message and the **ECIMouthData** type.

Synthesis State Parameter Defaults

The following table provides a summary of the synthesis state parameters and their default behavior.

Parameter	Default value	Default behavior
eciDictionary	0	User dictionaries are used.
eciInputType	0	Annotations in input will be spelled out.
eciLanguageDialect	lowest number installed	The lowest-numbered language/dialect on the system is used.
eciNumberMode	1	Four-digit numbers are pronounced as “years”.
eciNumParams	0	Total number of ECIParams.
eciRealWorldUnits	0	ECI units are used for all voice definition parameters.
eciSampleRate	1	The sample rate is 11,025 samples per second.
eciSynthMode	0	The input buffer is synthesized and cleared at the end of each sentence.
eciTextMode	0	No special spelling interpretation is performed on the text.
eciWantPhonemeIndices	0	Phoneme indices are not generated.

Voice Parameters

Voice parameters are commands used to define and adjust individual voice characteristics. A set of voice parameters makes a voice definition. You can create custom voices by selecting unique combinations of voice parameters. In addition, there are five predefined voice definitions, as discussed in the next section.

When you create a new ECI instance, it is given the default voice parameters. You can:

- Get the current voice parameters using **eciGetVoiceParam**.
- Set the voice parameters directly through **eciSetVoiceParam**, or indirectly by sending annotated text in calls to **eciAddText**.

This section describes the voice parameters that can be passed to **eciGetVoiceParam** and **eciSetVoiceParam**.

eciBreathiness

Range: 0-100

This parameter controls the amount of breathiness in the voice. The higher the value, the more breathiness the voice has. A value of 100 produces a whisper.

This voice parameter can be changed using the annotation ‘vyN (see Selecting a Voice or Voice Characteristics).

eciGender

0: male
1: female

Male and female vocal tracts have physical differences that affect the voice, some of which are reflected in the vocal tract setting. Other differences between male and female voices, namely pitch and head size, are controlled independently.

This voice parameter can be changed using the annotation ‘vgN (see Selecting a Voice or Voice Characteristics).

eciHeadSize

Range: 0-100

This parameter controls the size of the head for the speaker, changing the perceived pitch and other acoustic characteristics of the voice. A large number indicates a large head and a deeper voice.

This voice parameter can be changed using the annotation ‘vhN (see Selecting a Voice or Voice Characteristics).

eciNumVoiceParams

Total number of ECIVoiceParams. Passing eciNumVoiceParams to eciGetVoiceParam will cause a -1 (error) return, which is an expected behavior.

eciPitchBaseline

Range: 0-100 (ECI units); **40-422** (Real World Units = cycles per second)

Changing the pitch baseline will affect the overall pitch of the voice. The larger the pitch value, the higher the pitch of the voice.

This voice parameter can be changed using the annotation ‘vbN (see Selecting a Voice or Voice Characteristics).

eciPitchFluctuation

Range: 0-100

This parameter controls the degree of pitch fluctuation in the voice. A value of zero produces a voice with no pitch fluctuation, resulting in monotone speech. A high value produces a voice with large pitch fluctuations, typical of excited speech.

This voice parameter can be changed using the annotation ‘vfN (see Selecting a Voice or Voice Characteristics).

eciRoughness

Range: 0-100

This parameter adds roughness or "creakiness" to the voice. A low value produces a smooth voice, while a high value is rough or scratchy.

This voice parameter can be changed using the annotation ‘vrN (see Selecting a Voice or Voice Characteristics).

eciSpeed

Range: 0-250 (ECI Units)**; 70-1297** (Real World Units = words per minute)

Speed controls the number of words spoken per minute.

This voice parameter can be changed using the annotation ‘vsN (see Selecting a Voice or Voice Characteristics).

eciVolume:

Range: 0-100 (ECI Units)**; 1-65535** (Real World Units)

The smaller the value, the lower the volume. Louder settings may cause distortion when combined with other attribute changes.

This voice parameter can be changed using the annotation ‘vvN (see Selecting a Voice or Voice Characteristics).

Preset Voice Definitions

Voice definitions are sets of parameter values that make an individual voice. There are five preset voice definitions for each dialect of each language (three more are reserved for future use).

Each voice definition contains a set of parameter values that control the attributes of the voice.

The preset voices in each language are:

1. Adult Male 1
2. Adult Female 1
3. Child 1
4. Adult Male 2
5. Adult Male 3
6. Adult Female 2
7. Elderly Female 1
8. Elderly Male 1

Voice Parameter Defaults

The following chart shows the voice definition parameters for all languages, except as noted:

	1	2	3	4	5	6	7	8
Voice Parameters	**Adult Male 1**	**Adult Female 1**	**Child 1**	**Adult Male 2**	**Adult Male 3**	**Adult Female 2**	**Elderly Female 1**	**Elderly Male 1**
Breathiness	0**	50	0	0	0	40	40	20
Gender	0	1	1	0	0	1	1	0
Head size	50	50	22	86	50	56	45	30

	1	2	3	4	5	6	7	8
Voice Parameters	**Adult Male 1**	**Adult Female 1**	**Child 1**	**Adult Male 2**	**Adult Male 3**	**Adult Female 2**	**Elderly Female 1**	**Elderly Male 1**
Pitch Baseline (ECI units)	65*	81	93	56	69	89	68	61
Pitch Fluctuation	30	30	35	47	34	35	30	44
Roughness	0	0	0	0	0	0	3	18
Speed (ECI units)	50	50	50	50	70	70	50	50
Volume (ECI units)	92	100	90	93	92	95	90	90

*In French, the Pitch Baseline parameter is 69. ** In Taiwanese Mandarin, the Breathiness paramter is 34.

Table of Functions

This table outlines the available ECI functions. Detailed information about each function can be found in the Alphabetical Index of Functions.

System Control

Use the following functions for system control:

Function	Description
eciDeactivateFilter	Disables the specified filter for the ECI instance.
eciNew	Creates a new ECI instance and returns a handle to it.
eciNewEx	Creates a new instance of ECI and returns a handle to it. The client indicates the language, dialect and character set for the new engine instance
eciReset	Resets the ECI instance to the default state.
eciSpeakText	Synthesizes text to the default audio device.
eciSpeakTextEx	Synthesizes text to the default audio device with ability for selection of language dialect and character set of its text

Synthesis Control

Use the following functions for Synthesis Control:

Function	Description
eciAddText	Appends new text to the input buffer.
eciClearInput	Clears the input buffer.
eciGeneratePhonemes	Converts text to phonemes.
eciGetIndex	Returns the last index reached in an output buffer.
eciInsertIndex	Inserts an index into an input buffer.
eciPause	Pauses or unpauses speech synthesis and playback.

Function	Description
eciSpeaking	Determines whether synthesis is in progress.
eciStop	Stops synthesis.
eciSynchronize	Waits for an ECI instance to finish processing its output and then synchronizes it with a device.
eciSynthesize	Starts synthesis of text in an input buffer.
eciSynthesizeFile	Synthesizes the contents of a file.

Output Control

Use the following functions for output control:.

Function	Description
eciSetOutputBuffer	Sets an output buffer as the synthesis destination.
eciSetOutputDevice	Sets an audio output hardware device as the synthesis destination.
eciSetOutputFilename	Sets an output file as the synthesis destination.

Speech Environment Parameter Selection

Use the following functions for speech environment parameter selection:

Function	Description
eciGetDefaultParam	Returns the default values for an environment speech parameter.
eciGetParam	Returns the value of an environment parameter.
eciSetDefaultParam	Sets the default values for an environment speech parameter.
eciSetParam	Sets an environment parameter.

Voice Parameter Control

Use the following functions for voice parameter control:

Function	Description
eciCopyVoice	Makes a copy of a set of voice parameters.
eciGetVoiceName	Returns the voice name and then copies it to a name buffer.
eciGetVoiceParam	Returns a voice parameter.
eciSetVoiceName	Sets a voice parameter.

Dynamic Dictionary Maintenance

Use the following functions for dictionary maintenance (for Asian languages such as Chinese and Japanese, use the functions that end with the letter A):

Function	Description
eciDeleteDict	Deletes a specified dictionary set.
eciDictFindFirst	Retrieves the first entry in a dictionary.
eciDictFindFirstA	Retrieves the first entry in a Chinese or Japanese dictionary.
eciDictFindNext	Retrieves the next entry in a dictionary.
eciDictFindNextA	Retrieves the next entry in a Chinese or Japanese dictionary.
eciDictLookup	Returns a pointer to the translation value for a key.
eciDictLookupA	Returns a pointer to the Chinese or Japanese translation value for a key.
eciGetDict	Returns a handle to an active dictionary set.
eciLoadDict	Loads a dictionary volume.
eciNewDict	Creates a new dictionary set for a given ECI handle.
eciSaveDict	Writes the contents of a dictionary volume to a file.

Function	Description
eciSetDict	Sets a dictionary set as the current dictionary set for a given ECI instance and the active language.
eciUpdateDict	Updates a dictionary volume with a key/translation pair.
eciUpdateDictA	Updates a Chinese or Japanese dictionary with a key/translation pair.

Diagnostics

Use the following table for diagnostics:

Functions	Description
eciClearErrors	Clears error bits.
eciErrorMessage	Returns an error message describing the last error encountered.
eciProgStatus	Returns a set of error-reporting bits.
eciTestPhrase	Synthesizes a test phrase.
eciVersion	Returns the IBM TTS version number.

Callback

Use the following function to register callbacks:

Function	Description
eciRegisterCallback	Registers a callback function with the ECI instance.

Custom Filters

Use the following functions to use custom filters:

Function	Description
eciDeactivateFilter	Disables the specified filter for the ECI instance.
eciDeleteFilter	Deletes a specified filter handle, deactivating all transformation performed by this filter if it is active, and freeing all resources used by the filter.
eciGetFilteredText	Returns the resulting filtered text for the input string processed by the specified filter. This function allows client applications to determine the text that will be sent to the synthesis engine after filtering.
eciNewFilter	Creates a new instance of an ECI Filter and returns a handle to it.
eciActivateFilter	Enables the specified filter for the ECI instance.
eciUpdateFilter	Allow runtime update of the token replacement applied to known fields.

Alphabetical Index of Functions

Following is an alphabetical description of the syntax and semantics of all ECI functions. Refer to the Table of Functions to find the function names associated with specific operations.

Parameters

Unless otherwise specified, valid ECIHand parameters are assumed to be non-NULL (not equal to NULL_ECI_HAND), and all pointers are assumed to be non-NULL. All strings are in ASCIIZ format.

Calling Conventions

On Win32 platforms, ECI functions are defined as __stdcall, formerly known as the PASCAL calling convention. Refer to the *Microsoft Visual C++ Programmers' Guide* for more information about this convention.

On UNIX platforms, ECI functions default to ordinary C functions.

eciActivateFilter

Enables the specified filter for the ECI instance.

Syntax

```
ECIFilterError eciActivateFilter (
  ECIHand       hEngine,
  ECIFilterHand whichFilterHand
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

whichFilterHand
Handle to indicate the ECI Filter Instance that is going to be transforming the text. This is the value returned by eciNewFilter.

Return Values

ECIFilterError
One of the values enumerated in type **ECIFilterError**. See Data Types for this enumeration.

Remarks

Multiple filter can be active at the same time.

See Also

eciNew, eciNewEx, Custom Filters

eciAddText

Appends new text to the input buffer.

Syntax

```
Boolean eciAddText(
  ECIHand eciHandle,
  ECIInputText text
);
```

Parameters

eciHandle
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

text
: Non-NULL pointer to the text to be synthesized (a null-terminated C-string).

Return Values

true
: A copy of your text was added to the input buffer.

false
: Failure. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

Appends new text to the end of the input buffer. When synthesized, the newly-added text will be spoken with the voice definition specified by the state of the active voice when the text is inserted.

You can add more text while the engine is still synthesizing previously added text. Sentences may be split between calls to **eciAddText**, but words may not. See **eciSynthesize** below for more information on sentence parsing.

Example

```
#include <stdio.h>
#include "eci.h"

//print a string to stdout and wait for any key
void showMessage( char *msg )
{
  printf( msg );
  getchar();
}

int main( int argc, char *argv[] )
{
  ECIHand myECI;
  FILE *myFP;
  char errorMsg[100];
  myECI = eciNew(); // create a new ECIHand
  if ( NULL_ECI_HAND == myECI )
    showMessage( "eciNew failed.\n" );
  else
  {
    if ( NULL != (myFP = fopen( argv[1], "rt" )) )
    {
      char buffer[1000];
      eciSetParam( myECI, eciSynthMode, 1 ); //set manual mode
      while( fgets(buffer, 1000, myFP) ) //read entire file
      {
        if ( !eciAddText(myECI, buffer) )
        {
          eciErrorMessage(myECI, errorMsg);
          showMessage( errorMsg );
        }
      }
```

continued on next page

```
    eciSynthesize( myECI ); //start synthesis
    eciSynchronize( myECI ); //wait for synthesis complete
    fclose( myFP );
   }
   eciDelete( myECI );//clean up
  }
}
```

See Also

eciSynthesize, eciSynchronize, eciSetParam, eciSetVoiceName, eciCopyVoice, eciErrorMessage, eciProgStatus

eciClearErrors

Resets error-reporting bits.

Syntax

```
void eciClearErrors(
  ECIHand hEngine,
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

None.

See Also

eciErrorMessage, eciProgStatus

eciClearInput

Clears the input buffer. Does not abort any synthesis already in progress.

Syntax

```
Boolean eciClearInput(
  ECIHand hEngine
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

true
The input buffer has been cleared.

false
An error occurred.

Remarks

The input buffer can be cleared only if the ECI instance is in manual mode. In automatic mode, the input buffer is transferred immediately to the synthesis engine and cannot be cleared.

Other functions that clear the input buffer are: **eciDelete**, **eciReset**, and **eciStop**.

When this function succeeds, text that has been added to the input buffer in manual mode is removed from the buffer unless it has already been sent to the engine. All resources associated with the input buffer are returned to the system.

Example

```
#include <stdio.h>
#include "eci.h"

//print a string to stdout and wait for any key
void showMessage( char *msg )
{
  printf( msg );
  getchar();
}

int main( int argc, char *argv[] )
{
  ECIHand myECI;
  FILE *myFP;
  char errorMsg[100];

  myECI = eciNew();//create a new ECIHand
  if ( NULL_ECI_HAND == myECI )
    showMessage( "eciNew failed.\n" );
  else
  {
    if ( NULL != (myFP = fopen( argv[1], "rt" )) )
    {
      char buffer[1000];
      eciSetParam( myECI, eciSynthMode, 1 ); //set manual mode
      while( fgets(buffer, 1000, myFP) )//read entire file
      {
        if ( !eciAddText(myECI, buffer) )
        {
          eciErrorMessage(myECI, errorMsg);showMessage( errorMsg );
          showMessage( errorMsg );
        }
      }
```

continued on next page

```
continued from previous page
    if ( ferror(myFP) )
    {
      showMessage( "Error reading input file\n" );
      if ( !eciClearInput(myECI) )
        showMessage( "Error clearing input buffer\n" );
    }
    else
    {
      eciSynthesize( myECI ); //start synthesis
      eciSynchronize( myECI ); //wait for synthesis complete
    }
    fclose( myFP );
  }
  eciDelete( myECI ); //clean up
 }
}
```

See Also

eciDeactivateFilter, eciReset, eciStop

eciCopyVoice

Makes a copy of a set of voice parameters.

Syntax

```
Boolean eciCopyVoice(
  ECIHand hEngine,
  int voiceFrom,
  int voiceTo
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

voiceFrom
: Voice to copy. You can copy a preset voice (1-3, 7-8), a user-defined voice (9-16), or the active voice (0).

voiceTo
: Voice to store the copy of *voiceFrom*. Either 0 (the active voice), or 9–16 (a user-defined voice).

Return Values

true
: The voice was copied.

false
: Failure. Parameter may be out of range.

Remarks

A “voice” is a set of voice parameters. Voice 0 indicates the active voice. When you add text to the input buffer, it is synthesized with voice 0.

Each IBM Text-to-Speech language comes with five preset voices. The default voice is voice 1. When you create a new ECI instance, voice 1 is automatically copied to voice 0 and becomes the active voice. The user-defined voices are initially undefined.

You can change the parameters of the active voice, and of the user-defined voices, by calling **eciSetVoiceParam**. You cannot change any of the preset voices with **eciSetVoiceParam**. If you want to change any of the preset voices, then you must first use **eciCopyVoice** to copy it to either the active voice or one of the user-defined voices.

Example

```
#include <stdio.h>
#include "eci.h"

//print a string to stdout and wait for any key
void showMessage( char *msg )
{
  printf( msg );
  getchar();
}

continued on next page
```

```
continued from previous page
int main( int argc, char *argv[] )
{
  ECIHand myECI;
  int voice;
  char buffer[64];

  myECI = eciNew();
  //create a new ECIHand
  if ( NULL_ECI_HAND == myECI )
    showMessage( "eciNew failed.\n" );
  else
  {
    eciAddText(myECI, "Default voice." );
    for( voice = 1; voice <= ECI_PRESET_VOICES; voice++ )
    {
      if ( !eciCopyVoice(myECI, voice, 0 ) )
      {
        sprintf( buffer, "Cannot not copy voice %d to 0\n", voice );
        showMessage( buffer );
      }
      else
      {
        sprintf( buffer, "Preset voice %d.", voice );
        eciAddText( myECI, buffer );
      }
    }
    eciSynthesize( myECI ); //start synthesis
    eciSynchronize( myECI );//wait for synthesis complete
    eciDelete( myECI );//clean up
  }
}
```

See Also

eciSetVoiceName, eciGetVoiceParam, eciGetVoiceName

eciDeactivateFilter

Disables the specified filter for the ECI instance.

Syntax

```
ECIFilterError eciDeactivateFilter (
  ECIHand       hEngine,
  ECIFilterHand whichFilterHand
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

whichFilterHand
Handle to indicate the ECI Filter Instance to disable. This is the value returned by eciNewFilter.

Return Values

ECIFilterError
One of the values enumerated in type **ECIFilterError**. See Data Types for this enumeration.

Remarks

See Also

eciActivateFilter, eciNew, eciNewEx, Custom Filters

eciDelete

Terminates synthesis and deletes the ECI instance.

Syntax

```
ECIHand eciDelete(
  ECIHand hEngine
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

NULL_ECI_HAND
The ECI instance was successfully destroyed.

Remarks

This function closes and returns to the system all resources associated with this ECI instance, including memory, handles, etc. Any synthesis which is underway when this function is called it is immediately terminated.

Example

```
#include <stdio.h>
#include "eci.h"

void showMessage( char *msg )
{
  printf( msg );
  getchar();
}

int main( int argc, char *argv[] )
{
  ECIHand myECI;

  myECI = eciNew();
  if ( NULL_ECI_HAND == myECI )
    showMessage( "eciNew failed.\n" );
  else
  {
    showMessage( "eciNew succeeded!\n" );
    eciAddText( myECI, "This is a test." );
    eciSynthesize( myECI );
    eciSynchronize( myECI );
    eciDelete( myECI );
  }
}
```

See Also

eciNew, eciStop, eciReset

eciDeleteDict

Deletes a specified dictionary set, deactivating all dynamic dictionary lookups for this ECI instance.

Syntax

```
ECIDictHand eciDeleteDict(
  ECIHand hEngine,
  ECIDictHand dictHandle
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

dictHandle
: Handle to the dictionary set to be deleted.

Return Values

NULL_DICT_HAND
: The requested dictionary set was successfully deleted.

See Also

eciNewDict, eciGetDict

eciDeleteFilter

Deletes a specified filter handle, deactivating all transformation performed by this filter if it is active, and freeing all resources used by the filter.

Syntax

```
ECIFilterHand eciDeleteFilter (
  ECIHand hEngine,
  ECIFilterHand whichFilterHand
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

whichFilterHand
Handle to the filter to be deleted. This is the value returned by eciNewFilter.

Return Values

ECIFilterHand
NULL_FILTER_HAND.

Remarks

See Also

eciNewFilter, eciNew, eciNewEx, Custom Filters

eciDictFindFirst

Retrieves the first entry in a dictionary volume.

Syntax

```
ECIDictError eciDictFindFirst(
  ECIHand hEngine,
  ECIDictHand dictHandle,
  ECIDictVolume whichDictionary,
  ECIInputText *ppKey,
  ECIInputText *ppTranslationValue
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

dictHandle
Handle to the dictionary set.

whichDictionary
One of the values enumerated in type **ECIDictVolume**. See Data Types for this enumeration.

ppKey
Pointer to the address of the key to the first entry in this dictionary. The key is a constant C-string.

pptranslationValue
Pointer to the address of the translation value of the first entry in this dictionary. The translation value is a constant C-string.

Return Values

ECIDictError
One of the values enumerated in type ECIDictError. See ECIDictError in Data Types for this enumeration.

Remarks

Retrieves the first dictionary entry. The ppK*ey* and *ppTranslationValue* will receive pointers to their corresponding strings in the dictionary. These should not be modified (or deallocated), as the dictionary may become corrupted and synthesis may fail. *ECIDictError* indicates whether any errors occurred in the call to **eciDictFindFirst**.
Refer to the section on User Dictionaries for more information about the ppK*ey* and ppT*ranslationValue* parameters.

See Also

eciNewDict, eciSetDict, eciDictFindNext, eciUpdateDict, User Dictionaries, Symbolic Phonetic Representations

eciDictFindFirstA

Retrieves the first entry in a dictionary volume. This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages.

Syntax

```
ECIDictError eciDictFindFirstA(
  ECIHand hEngine,
  ECIDictHand dictHandle,
  ECIDictVolume whichDictionary,
  ECIInputText *ppKey,
  ECIInputText *ppTranslationValue
  ECIPartOfSpeech *pPartOfSpeech
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

dictHandle
Handle to the dictionary set.

whichDictionary
Volume of the dictionary set. This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages.

ppKey
Pointer to the address of the key to the first entry in this dictionary. This is the address of the variable where a pointer to a constant buffer is returned.

ppTranslationValue
Pointer to the address of the translation value of the first entry in this dictionary. This is the address of the variable where a pointer to a constant buffer is returned.

pPartOfSpeech
Pointer to the ECIPartOfSpeech enumeration, which specifies the grammatical category.

Return Codes

ECIDictError

One of the values enumerated in type ECIDictError. See ECIDictError in Data Types for this enumeration.

Remarks

This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages.This function starts scanning through the dictionary from the beginning and retrieves the first entry. The ppKey and ppTranslationValue parameters receive pointers to their corresponding strings in the dictionary. These should neither be modified nor deallocated because the dictionary can become corrupted, causing speech synthesis to fail. ECIDictError indicates whether any errors occurred in the call to **eciDictFindFirstA**.

The buffer contents should be in the same code page currently selected for this speech synthesis engine instance. If a Unicode code page is active, *ppKey* and *ppTranslationValue* should be in wide-character (Unicode) format with a 16-bit terminator. Otherwise, *ppKey* and *ppTranslationValue* should be an 8-bit, NULL-terminated C string.

See Also

eciDeleteDict, eciDictFindFirstA,eciDictFindNextA, eciDictLookupA , eciLoadDict, eciSaveDict, eciSetDict, eciUpdateDictA

eciDictFindNext

Retrieves the next dictionary entry following the last entry retrieved.

Syntax

```
ECIDictError eciDictFindNext(
  ECIHand hEngine,
  ECIDictHand dictHandle,
  ECIDictVolume whichDictionary,
  ECIInputText *ppKey,
  ECIInputText *ppTranslationValue
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

dictHandle
Handle to the dictionary set.

whichDictionary
One of the values enumerated in type **ECIDictVolume**. See Data Types for this enumeration.

ppKey
Pointer to the address of the key to the next entry in this dictionary. The key is a constant C string.

ppTranslationValue
Pointer to the address of the translation value of the next entry in this dictionary. The translation value is a constant C string.

Return Values

ECIDictError
One of the values enumerated in type ECIDictError. See ECIDictError in Data Types for this enumeration.

Remarks

The input and output parameters have the same meaning as they do for the **eciDictFindFirst** function. The first call to this function should be preceded by a call to **eciDictFindFirst**. Entries are not returned in any particular order. An **ECIDictError** is returned which indicates any errors that occurred in the call to **eciDictFindNext**. **DictNoEntry** is returned if there are no more entries in the dictionary.

Parameter and return code enumerations are declared in eci.h.

See Also

eciNewDict, eciDictFindFirst, eciUpdateDict

eciDictFindNextA

Retrieves the next dictionary entry following the last entry retrieved. This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages.

Syntax

```
ECIDictError eciDictFindNextA(
  ECIHand hEngine,
  ECIDictHand dictHandle,
  ECIDictVolume whichDictionary,
  ECIInputText *ppKey,
  ECIInputText *ppTranslationValue
  ECIPartOfSpeech *pPartOfSpeech
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

dictHandle
Handle to the dictionary set.

whichDictionary
Volume of the dictionary set. This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages.

ppKey
Pointer to a key to the next entry in this dictionary. This is the address of the variable where a pointer to a constant buffer is returned.

ppTranslationValue
Pointer to the value of the next entry in this dictionary. This is the address of the variable where a pointer to a constant buffer is returned.

pPartOfSpeech
Pointer to the **ECIPartOfSpeech** enumeration, which specifies the grammatical category.

Return Values

ECIDictError

One of the values enumerated in type ECIDictError. See ECIDictError in Data Types for this enumeration.

Remarks

This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages.This function retrieves the next entry in the dictionary. The input and output parameters have the same meaning as they do for **eciDictFindFirstA**. This function returns DictNoEntry if there are no more entries in the dictionary. The first call to this function should be preceded by a call to **eciDictFindFirstA**. Entries are not returned in any particular order. This function returns ECIDictError if any errors occurred in the call to **eciDictFindNextA**.

The buffer contents should be in the same code page currently selected for this speech synthesis engine instance. If a Unicode code page is active, *ppKey* and *ppTranslationValue* should be in wide-character (Unicode) format with a 16-bit terminator. Otherwise, *ppKey* and *ppTranslationValue* should be an 8-bit, NULL-terminated C string.

Parameter and return code enumerations are declared in eci.h.

See Also

eciDeleteDict, eciDictFindFirstA,eciDictFindNextA, eciDictLookupA , eciLoadDict, eciSaveDict, eciSetDict, eciUpdateDictA

eciDictLookup

Returns a pointer to the translation value for *key*.

Syntax

```
const char* eciDictLookup(
  ECIHand hEngine,
  ECIDictHand dictHandle,
  ECIDictVolume whichDictionary,
  ECIInputText key
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

dictHandle
: Handle to the dictionary set.

whichDictionary
: One of the values enumerated in type **ECIDictVolume**. See Data Types for this enumeration.

key
: Pointer to a key to the entry whose value you want. The key is a null-terminated C string.

Return Values

NULL
: The *key* is not in the dictionary.

non-NULL
: A pointer to the translation value for *key*.

Remarks

Returns a pointer to the translation value for *key* or NULL if the *key* is not in the dictionary. The string referenced by the return value should not be modified, as the dictionary may become corrupted and synthesis may fail. Parameter and return code enumerations are declared in `eci.h`.

See Also

eciNewDict, eciUpdateDict

eciDictLookupA

Returns a pointer to the translation value for *pkey.*This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages.

Syntax

```
ECIDictError eciDictLookupA(
  ECIHand hEngine,
  ECIDictHand hDict,
  ECIDictVolume DictVol,
  ECIInputText pKey,
  ECIInputText *ppTranslationValue,
  ECIPartOfSpeech *pPartOfSpeech
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by **eciNew** or **eciNewEx**.

hDict
: Handle to the dictionary set. This is the value returned by **eciNewDict** or **eciGetDict**.

DictVol
: Volume of the dictionary set. This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages.

pKey
: Pointer to a key to the entry whose value you want. This is a NULL-terminated buffer containing the key.

ppTranslationValue
: Pointer to the value of the next entry in this dictionary. This is the address of the variable where a pointer to a constant buffer is returned.

pPartOfSpeech
: Pointer to the ECIPartOfSpeech enumeration, which specifies the grammatical category.

Return Values

ECIDictError

One of the values enumerated in type ECIDictError. See ECIDictError in Data Types for this enumeration.

Remarks

This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages.This function returns a pointer to the translation value for *key*, or NULL if the key is not in the dictionary. The string referenced by the return value should not be modified because the dictionary can become corrupted, causing speech synthesis to fail.
The buffer contents should be in the same code page currently selected for this speech synthesis engine instance. If a Unicode code page is active, *pKey* and *ppTranslationValue* should be in wide-character (Unicode) format with a 16-bit terminator. Otherwise, *pKey* and *ppTranslationValue* should be an 8-bit, NULL-terminated C string.

See Also

eciDeleteDict, eciDictFindFirstA,eciDictFindNextA, eciDictLookupA , eciLoadDict, eciSaveDict, eciSetDict, eciUpdateDictA

eciErrorMessage

Copies an error message describing the last error encountered into the buffer as an ASCIIZ string.

Syntax

```
void eciErrorMessage(
  ECIHand eciHandle,
  void* buffer
);
```

Parameters

eciHandle

Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

buffer

Pointer to the buffer to receive the error message. This buffer must have room for at least 100 characters.

Return Values

None.

Remarks

Copies an error message describing the latest error into the buffer pointed to by *buffer*. The buffer will contain an empty string if there have been no errors. If *eciHandle* is NULL_ECI_HAND, copies a message about insufficient memory to *buffer*.

See Also

eciClearErrors, eciProgStatus

eciGeneratePhonemes

Converts text to phonemes.

Syntax

```
Boolean eciGeneratePhonemes(
  ECIHand eciHandle,
  int size,
  void* buffer
);
```

Parameters

eciHandle
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

size
Size of buffer (in bytes). 0 cancels phoneme generation

buffer
Pointer to the buffer to receive phonemes. NULL cancels phoneme generation.

Return Values

true
Phoneme generation was successfully performed.

false
Failure. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information. See remarks.

Remarks

The **eciSynthMode** must be set to manual using **eciSetParam** before text is added to the input buffer.

A callback should be registered with **eciRegisterCallback** before you call **eciGeneratePhonemes**. Synthesis may not be underway when this function is called.

The text in the synthesis engine's input buffer is converted to phonemes and placed in the designated phoneme buffer. When your buffer is full, or all the text has been converted, whichever comes first, your callback is called with an **eciPhonemeBuffer** message. If your phoneme buffer cannot hold all the generated phonemes, your callback is called repeatedly. You need to process the contents of your phoneme buffer every time your callback is called; otherwise, the contents may be overwritten as the input text continues to be converted to phonemes.

eciGeneratePhonemes returns synchronously when phoneme conversion is complete**.**

Common conditions that cause this function to return "false":

- **eciSynthMode** is set to *sentence*.
- No callback is registered.
- Synthesis is already underway.

See Also

eciAddText, eciSetParam, eciSpeaking, eciRegisterCallback, eciProgStatus, eciErrorMessage

eciGetAvailableLanguages

Returns and identifies the number of installed and available languages.

Syntax

```
int (ECILanguageDialect *paLangs,
   int *piNumLangs
);
```

Parameters

paLangs

Pointer to an array of installed languages. Each element in the array is of type ECILAnguageDialect enumeration (defined in eci.h).

piNumLangs

[On input] Pointer to the number of available elements in the paLangs array. If this number is less than the number of available languages, then paLangs contains only that number of languages (starting from the lowest-numbered language).

[On output] Pointer to how many elements were filled in. If the number is 0 on input, then the number on output is the size required to hold the array.

Return Values

0

Success.

ECI_PARAMETERERROR

An error occurred because of improper parameters.

Remarks

This function allows a developer to query the installed languages without incurring the overhead of loading a language to see if it is present. It is the caller's responsibility to manage the dynamic memory that is required to hold the array.

See Also

N/A

eciGetDefaultParam

Returns the default value for an environment speech parameter.

Syntax

```
int eciGetDefaultParam(
   ECIParam Param
);
```

Parameters

Param
: Parameter value taken from the existing ECIParam enumeration in eci.h. These are the same enumeration values that are used by eciSetParam and eciGetParam.

Return Values

>= 0
: The default Param value.

-1
: An error. Param is out of range.

Remarks

N/A

See Also

N/A

eciGetDict

Returns the handle to the active dictionary set for the current language.

Syntax

```
ECIDictHand eciGetDict(
  ECIHand hEngine
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

Non-null
A valid dictionary set handle.

NULL_DICT_HAND
There is no active dictionary set for this instance.

Remarks

This function returns the handle to the currently active dictionary set for the current language, or NULL_DICT_HAND, if there is no active set. The synthesis engine does not perform any dynamic dictionary lookups until a dictionary set is established as the current set using **eciSetDict**.

See Also

eciNewDict, eciSetDict

eciGetFilteredText

Returns the resulting filtered text for the input string processed by the specified filter. This function allows client applications to determine the text that will be sent to the synthesis engine after filtering.

Syntax

```
ECIFilterError eciGetFilteredText (
  ECIHand hEngine,
  ECIFilterHand whichFilterHand,
  ECIInputText input,
  ECIInputText* filteredText
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

whichFilterHand
Handle to the filter to be used. This is the value returned by eciNewFilter.

input
Text to apply filter to.

filteredText
The resulting text after the specified filter is applied. The value of *filteredText* is only valid until the next call to eciGetFilteredText or eciDeleteFilter.

Return Values

ECIFilterError
One of the values enumerated in type **ECIFilterError**. See Data Types for this enumeration.

Remarks

For static filters, this function must be called first on the text to be filtered. The resulting text can then be sent to the synthesis engine using eciAddText.

See Also

eciAddText, eciNewFilter, eciNew, eciNewEx, Custom Filters

eciGetIndex

Returns the last index reached in an output buffer.

Syntax

```
int eciGetIndex(
  ECIHand hEngine
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

0
: No indices have been encountered yet.

non-zero
: The last index encountered in the output buffer.

Remarks

Returns the last index reached in the output buffer, or 0 if no index has been encountered. All inserted indices must be nonzero integer values; thus, the return value of **eciGetIndex** is unambiguous.

See Also

eciInsertIndex

eciGetParam

Returns the value of an environment parameter.

Syntax

```
int eciGetParam(
  ECIHand hEngine,
  ECIParam eciParameter
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

eciParameter
One of the values enumerated in type **ECIParam**. See Data Types for this enumeration.

Return Values

>= 0
The eciParameter value.

-1
An error occurred. Parameter may be out of range.

Remarks

Gets the value of an environment parameter. Returns a value greater than or equal to 0 on success, or -1 on failure.

See Also

Synthesis State Parameters

eciGetVoiceName

Copies the voice name to a name buffer.

Syntax

```
Boolean eciGetVoiceName(
  ECIHand hEngine,
  int voiceNum,
  void* nameBuffer
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

voiceNum
Number of the voice whose name you want: 0, 1-8, 9-16.

nameBuffer
Pointer to the buffer where a null-terminated C string containing the name will be copied. Must be non-NULL.

Return Values

true
The text string was successfully copied to the *nameBuffer*.

false
An error occurred. Check for invalid or out of range parameter.

Remarks

Your *nameBuffer* should be ECI_VOICE_NAME_LENGTH + 1 bytes long. Otherwise, a long voice name may corrupt memory.

See Also

eciSetVoiceName, Voice Parameters

eciGetVoiceParam

Returns a voice parameter.

Syntax

```
int eciGetVoiceParam(
  ECIHand hEngine,
  int voiceNum,
  ECIVoiceParam voiceParameter
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

voiceNum
The number of the voice whose parameter value you want. 0, 1-8, 9-16

voiceParameter
One of the values enumerated in type **ECIVoiceParam**. See Data Types for this enumeration.

Return Values

>= 0
The voice parameter value you requested.

-1
Failure. Parameter may be out of range.

Remarks

Gets the specified voice parameter value for the specified voice. A voice of 0 indicates the active voice.

See Also

eciSetVoiceName, Voice Parameters

eciInsertIndex

Inserts an index into the input buffer.

Syntax

```
Boolean eciInsertIndex(
  ECIHand hEngine,
  int indexNum
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

indexNum
: Unique index number to be inserted. If it is not unique, you may later be unable to determine which index is being returned.

Return Values

true
: Index inserted successfully.

false
: Error inserting index. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

Appends an index, with the specified number, to the input buffer. After all the text prior to this index has been synthesized, an **eciIndexReply** message containing this index number is sent to your callback function. If you are synthesizing to an audio device, then the index reply message is sent at about the same time the text is being heard on the speakers.

Indices must be nonzero integer values.

If no callback has been registered, the index reply message cannot be sent. You can still query the latest index with **eciGetIndex**.

See Also

eciGetIndex, eciRegisterCallback, eciErrorMessage, eciProgStatus

eciLoadDict

Loads a dictionary volume.

Syntax

```
ECIDictError eciLoadDict(
  ECIHand whichECI,
  ECIDictHand whichDictHand,
  ECIDictVolume whichDictionay,
  const void* filename
);
```

Parameters

whichECI
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

whichDictHand
Handle to the dictionary set.

whichDictionary
One of the values enumerated in type **ECIDictVolume**. See Data Types for this enumeration.

filename
Pointer to a null-terminated C string containing the name of a dictionary file, which may contain a path.

Return Values

ECIDictError
One of the values enumerated in type ECIDictError. See ECIDictError in Data Types for this enumeration.

Remarks

Loads the dictionary volume identified by *whichDictionary* from the file named by *filename*. The input file consists of ASCII text with one dictionary entry on each line. Each input line contains a key and the corresponding translation value, separated by a tab character. See User Dictionaries for a full

discussion of valid keys and translation values within each type of dictionary. An **ECIDictError** is returned, which indicates any error that occurred in the call to **eciLoadDict**.

See Also

eciNewDict, eciSetDict, eciSaveDict, eciDeleteDict, eciDictFindFirst, eciDictFindNext, eciDictLookup

eciNew

Creates a new instance of ECI and returns a handle to it.

Syntax

```
ECIHand eciNew(
  void
);
```

Parameters

None.

Return Values

ECIHand
: Handle to an ECI instance. This same value must be used in all subsequent calls to this ECI instance, such as **eciAddText**, **eciSynthesize**, **eciSynchronize**, **eciDelete**, and so on.

NULL_ECI_HAND
: An unrecoverable error occurred. Usually, this is because ECI could not locate the synthesis engine.

Remarks

This is typically the first call you will make to the ECI API unless you are using **eciSpeakText**. It creates a new ECI instance with default attributes. See Synthesis State Parameter Defaults.

Example

```
#include <stdio.h>
#include "eci.h"

void showMessage( char *msg )
{
  printf( msg );
  getchar();
}

int main( int argc, char *argv[] )
{
  ECIHand myECI;

  myECI = eciNew();
  if ( NULL_ECI_HAND == myECI )
    showMessage( "eciNew failed.\n" );
  else
  {
    showMessage( "eciNew succeeded!\n" );
    eciAddText( myECI, "This is a test." );
    eciSynthesize( myECI );
    eciSynchronize( myECI );
    eciDelete( myECI );
  }
}
```

See Also

eciAddText, eciSynchronize, eciDeactivateFilter, Synthesis State Parameters

eciNewEx

Creates a new instance of ECI and returns a handle to it. The client indicates the language, dialect and character set for the new engine instance.

Syntax

```
ECIHand eciNewEx(
  ECILanguageDialect value
);
```

Parameters

value
Value to indicate the ECI language dialect.

Return Values

ECIHand
Handle to one instance of the synthesis engine. This same value must be used in all subsequent calls to this instance of the synthesis engine, such as **eciAddText**, **eciSynthesize**, **eciSynchronize**, **eciDelete**, and so on.

NULL_ECI_HAND
An unrecoverable error occurred.

Remarks

This is typically the first call you will make to the ECI API. It creates a new ECI instance with default attributes for the language specified as argument.

Example

```
#include <stdio.h>
#include "eci.h"

void showMessage( char *msg )
{
  printf( msg );
  getchar();
}

int main( int argc, char *argv[] )
{
  ECIHand myECI;

  myECI = eciNewEx();
  if ( NULL_ECI_HAND == myECI )
    showMessage( "eciNew failed.\n" );
  else
  {
    showMessage( "eciNewEx succeeded!\n" );
    eciAddText( myECI, "This is a test." );
    eciSynchronize( myECI );
    eciDelete( myECI );
  }
}
```

See Also

eciAddText, eciSynchronize, eciDeactivateFilter, Synthesis State Parameters

eciNewDict

Creates a new dictionary set for a given ECI instance.

Syntax

```
ECIDictHand eciNewDict(
  ECIHand hEngine
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

ECIDictHand
: Handle to the dictionary set. Use this handle in subsequent dictionary calls that require an eciDictHand handle.

NULL_DICT_HAND
: The dictionary set could not be created.

Remarks

Creates a new, empty dictionary set for the given ECI instance and the current language. Use this function to create different dictionary handles for different languages by setting the language parameter using **eciSetParam** before each call to **eciNewDict**. IBM TTS will not look up entries in the new dictionary until it is activated with a call to **eciSetDict**. Returns NULL_DICT_HAND if the dictionary set could not be created.

See Also

eciLoadDict, eciDeleteDict, User Dictionaries

eciNewFilter

Creates a new instance of an ECI Filter and returns a handle to it.

Syntax

```
ECIFilterHand eciNewFilter (
  ECIHand       hEngine,
  unsigned int  filterNum
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

filterNum
Value indicating the numerical designation for this filter. If no value is specified, the default filter (number 0) is used.

Return Values

ECIFilterHand
Handle to an ECI Filter instance. This same value must be used in all subsequent calls to use this ECI Filter instance, such as, **`eciDeactivateFilter, eciActivateFilter, eciDeleteFilter, eciUpdateFilter,`** and **`eciGetFilteredText.`**

NULL_ECI_HAND
An unrecoverable error occurred.

Remarks

If no `filterNum` parameter is specified, the default filter is used (filter number 0). Multiple filter handles may be created for each ECI instance; also, multiple filters may be active at a time.

See Also

eciNew, eciNewEx, Custom Filters

eciPause

Pauses or unpauses speech synthesis and playback.

Syntax

```
Boolean eciPause(
  ECIHand hEngine,
  Boolean fPauseon
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

fPauseon
Boolean value that indicates whether to pause (`true`) or resume (`false`).

Return Values

true
Successfully set pause status as indicated in *fPauseon*.

false
Failure. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

If the variable *fPauseon* is set to `true`, the synthesis engine and the output device are paused. During a pause, no output is sent to the audio device or to your callback function. If the variable *fPauseon* is set to `false`, synthesis resumes where it left off.

See Also

eciProgStatus, eciErrorMessage

eciProgStatus

Returns a set of error-flag bits.

Syntax

```
Boolean eciProgStatus(
  ECIHand hEngine
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

0
No error.

Non-zero values are returned as an integer, composed of one or more of the following constants, joined by bitwise "or":

ECI_SYSTEMERROR
Operating system returned an error.

ECI_MEMORYERROR
System resources low.

ECI_MODULELOADERROR
Unable to load necessary program module.

ECI_DELTAERROR
Error in Delta program.

ECI_SYNTHERROR
Error in synthesis engine.

ECI_DEVICEERROR
Error using sound device.

ECI_PARAMETERERROR
: Invalid or out of range parameter.

ECI_SYNTHESIZINGERROR
: Synthesis engine is busy.

ECI_DEVICEBUSY
: Audio device is busy.

ECI_SYNTHESISPAUSED
: Synthesis engine is paused.

Remarks

All bits are cleared by **eciReset** and **eciClearErrors**. The last error set can be retrieved by calling **eciErrorMessage**.

See Also

eciReset, eciClearErrors, eciErrorMessage

eciRegisterCallback

Registers your callback function with an ECI instance.

Syntax

```
void eciRegisterCallback(
  ECIHand hEngine,
  ECICallback *pCallback
  void* data
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

pCallback
Pointer to the **ECICallback** function. Can be NULL.

data
Pointer to an arbitrary value, or a key to the size of a void pointer. All values are allowed.

Return Values

None.

Remarks

This function registers your callback function with an ECI instance. If *pCallback* is NULL, the current callback is removed. The supplied data pointer is associated with your callback. It is passed back to your callback function on entry, so that your function can use it, if desired, for identification purposes, such as a class pointer or an instance reference. Only one callback function can be registered at a time with each ECI instance. ECI functions may not be called from within a callback.

Your callback must be registered with **eciRegisterCallback** before any function that creates messages is called. The functions that cause messages to be sent to your callback are **eciGeneratePhonemes**,

eciInsertIndex, and **eciSetOutputBuffer**, and setting **eciWantPhonemeIndices** to 1 with **eciSetParam**.

For any given ECI instance, your callback will be called from the same thread on which your application calls ECI. This is achieved by passing control from your application to ECI. When you call **eciSynchronize**, ECI will retrieve all messages and execute your callback for each one, until synthesis is complete. If you call **eciSpeaking**, ECI will retrieve just those messages that are ready, execute your callback for each one, and then return. If you choose to use **eciSpeaking**, instead of **eciSynchronize**, you must keep calling it until it returns false.

eciRegisterCallback may not be called while synthesis is in progress.

The syntax of your callback is as follows:

```
ECICallbackReturn callback(
  ECIHand hEngine,
  ECIMessage msg,
  long lparam,
  void* data
);
```

Callback Function Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

msg
Enumeration indicating the type of message (see Data Types):

eciWaveformBuffer
eciPhonemeBuffer
eciIndexReply
eciPhonemeIndexReply

eciWordIndexReply

lparam
A long whose value and interpretation depends on the ECIMessage type. See discussion below.

data
An arbitrary value which is the size of a void pointer. You specify this value in your call to **eciRegisterCallback**. All values are allowed, including pointers.

Callback Function Return Values

eciDataProcessed
: You have processed the message and any associated data in your output buffer. Subsequent messages may be sent to your callback.

eciDataNotProcessed
: You could not process the message or associated data in your output buffer. The same message will be sent to your callback later.

If your callback processes the **ECIMessage**, and does not wish to see that same message again, it should return **eciDataProcessed**. If your callback function cannot process the message, and would like to see the same message again, it should return **eciDataNotProcessed**; your callback will be called with the same message at a later time. This is particularly useful if an **eciWaveformBuffer** message cannot be processed because the buffer you are writing to is temporarily full. No new **ECIMessage** will be sent if **eciDataNotProcessed** is returned. If your application continues to return **eciDataNotProcessed**, synthesis will stop, aslo, **eciDataAbort** will stop sythesis and clear the text buffer.

All callbacks should return quickly to ensure that there is no interruption of output. The value and interpretation of *lparam* is dependent on ECIMessage.

ECIMessage eciWaveformBuffer

lparam indicates the number of samples (not bytes) that have just been added to your output buffer. Your output buffer is specified in a call to **eciSetOutputBuffer**.

When phoneme indices are also being generated, this message is sent for the samples for each phoneme.

Samples are 16-bit signed PCM values and are centered at 0.

Once your callback returns eciDataProcessed, the data in your output buffer is no longer protected; therefore, your callback should only return eciDataProcessed when it has processed all the data in your buffer. No more data will be added to your buffer until eciDataProcessed is returned.

ECIMessage eciPhonemeBuffer

The *lparam* parameter indicates the number of characters (bytes) that have just been added to your phoneme buffer. Your phoneme buffer address was given to the ECI instance in your call to **eciGeneratePhonemes**.

Once your callback returns **eciDataProcessed**, the data in your phoneme buffer is no longer protected; therefore, your callback should only return **eciDataProcessed** when it has processed all the data in the buffer. More data will not be added to your phoneme buffer until **eciDataProcessed** is returned.

ECIMessage eciIndexReply

lparam is an index that was reached during synthesis and playback of the input text buffer. Indices are inserted into the input text buffer with **eciInsertIndex**.

Your callback should return immediately to ensure that there is no interruption of output.

Receiving index notifications is useful for synchronizing text with user-defined events, for example, word highlighting or simultaneous display of related graphics in a slideshow-style presentation.

ECIMessage eciPhonemeIndexReply

lparam is a pointer to an **ECIMouthData** structure.

This message is sent only when the eciWantPhonemeIndices environment parameter is set to 1. One of these messages is sent for each phoneme spoken, just before the phoneme starts playing on the audio device (or just before the associated waveform audio is placed in your output buffer, if you have called **eciSetOutputBuffer**).

In addition to the language-specific phoneme symbols, the symbol ¤ (0xA4) is used to indicate the end of an utterance, and is sent with a set of neutral mouth position parameters.

Receiving phoneme notifications this way is appropriate for synchronizing facial animation or other graphics with the speech output. If your application only needs to synchronize with individual words or larger units, use word indices (**eciIndexReply** messages). To convert text to phonemes, use the **eciGeneratePhonemes** function and the **eciPhonemeBuffer** message will be sent.

As with other messages, your callback function must return quickly. If significant processing is required, as with complex graphics, your application should spawn a new thread, and marshal the

callback messages to the new thread. Your application is responsible for skipping messages, if it receives them faster than it can process them.

Examples

The following example in C++ function converts an input string to phonemes and writes them to the console.

```
#include <eci.h>
#include <iostream.h>

const int bufferSize = 100;
static char buffer[bufferSize];

static ECICallbackReturn callback(ECIHand eciHand,
      ECIMessage msg, long lparam, void* data)
{
  if (msg == eciPhonemeBuffer)
  {
    cout << buffer;
  }
  return eciDataProcessed;
}

void showPhonemes(const char* text)
{
  //This function demonstrates the proper use of
  //eciGeneratePhonemes. The standard error checking of
  //ECI functions which return error codes is left out for
  //the sake of simplicity
  ECIHand eciHand = eciNew();

  eciRegisterCallback(eciHand, callback, 0);
  eciSetParam(eciHand, eciSynthMode,1);
  eciAddText(eciHand, text);
  eciGeneratePhonemes(eciHand, bufferSize, buffer);
  eciDelete(eciHand);
  return;
}
```

The following C++ example illustrates how to capture the waveform samples in order to process them using an alternate method, rather than sending them directly to the audio device:

```
#include <eci.h>
#include <iostream.h>

const int bufferSize = 100;
static short buffer[bufferSize];

// This function performs some kind of processing on the samples
// It returns true if it is done with those samples,
// and false if it wants to be called with the same samples again.
extern Boolean handleSamples(const short* samples, long count);

static ECICallbackReturn callback(ECIHand eciHand,
      ECIMessage msg, long lparam, void* data)
{
  Boolean retval = true;
  if (msg == eciWaveformBuffer)
  {
    retval = handleSamples(buffer, lparam);
  }
  return (retval?eciDataProcessed:eciDataNotProcessed);
}
void collectSamples(const char* text)
{
  //This function demonstrates the proper use of the
  //eciWaveformBuffer message. The standard error checking
  //of ECI functions which return error codes is left out
  //for the sake of simplicity

  ECIHand eciHand = eciNew();

  eciRegisterCallback(eciHand, callback, 0);
  eciSetOutputBuffer(eciHand, bufferSize, buffer);
  eciAddText(eciHand, text);
  eciSynthesize(eciHand);
  //Wait until synthesis is complete
  eciSynchronize (eciHand);
  eciDelete(eciHand);
}
```

The following C++ program uses word indices to synchronize a visible countdown with an audible one:

```
#include <iostream.h>
#include <stdio.h>
#include <eci.h>

static ECICallbackReturn callback(ECIHand eciHand,
      ECIMessage msg, long lparam, void* data)
{
  if (msg == eciIndexReply)
  {
    cout << param << "..." << endl;
  }
  return eciDataProcessed;
}

int main(int argc, const char* argv[])
{
  //This function demonstrates the proper use of the
  //eciIndexReply message. The standard error checking
  //of ECI functions which return error codes is left out
  //for the sake of simplicity

  ECIHand eciHand = eciNew();

  eciRegisterCallback(eciHand, callback, 0);
  // count from 10 to 0, inserting an index at each count
  for (int i = 10; i > 0; i--)
  {
    char buf[10];
    sprintf(buf, "%d, ", i);
    eciInsertIndex(eciHand, i);
    eciAddText(eciHand, buf);
  }
  eciInsertIndex(eciHand, 0);
  eciAddText(eciHand, "go!");
```

continued on next page

```
 continued from previous page
 // synthesize, and wait until speaking is finished
 eciSynthesize(eciHand);
 //Wait until synthesis is complete
 eciSynchronize (eciHand);
 cout << "Gone." << endl;

 eciDelete(eciHand);
 return 0;
}
```

Phoneme indices may be used in a similar way, except that there is no need for calls to **eciInsertIndex**. The callback function might look like this:

```
static ECICallbackReturn callback(ECIHand eciHand,
      ECIMessage msg, long lparam, void* data)
{
 if (msg == eciPhonemeIndexReply)
 {
  const ECIMouthData* mouthData = (ECIMouthData*)lparam;

  // process the mouth position data--here, we'll just
  // print the phoneme name
  cout << mouthData->phoneme << endl;
 }
 return eciDataProcessed;
}
```

See Also

eciSynchronize, eciSpeaking, Data Types

eciReset

Resets the ECI instance to the default state.

Syntax

```
Boolean eciReset(
  ECIHand hEngine
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

true
: Success

false
: Failure. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

The default state is the same the state of a new instance returned by **eciNew**. See Synthesis State Parameter Defaults.

If synthesis is underway when this function is called, the synthesis is terminated immediately.

See Also

eciNew, Synthesis State Parameters, eciErrorMessage, eciProgStatus

eciSaveDict

Writes the contents of a dictionary volume to a file.

Syntax

```
ECIDictError eciSaveDict(
  ECIHand whichECI,
  ECIDictHand whichDictHand,
  ECIDictVolume whichDictionary,
  const void* filename
);
```

Parameters

whichECI.
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

whichDictHand
Handle to the dictionary set.

whichDictionary
One of the values enumerated in type **ECIDictVolume**. See Data Types for this enumeration.

filename
Pointer to the name of a dictionary file. This is a null-terminated C string containing the name of a dictionary file.

Return Values

ECIDictError
One of the values enumerated in type ECIDictError. See ECIDictError in Data Types for this enumeration.

Remarks

The ASCII file will be in a format suitable for reloading with **eciLoadDict**. The entries are listed in no particular order, and will generally be different from the order entered or loaded. An **ECIDictError** is returned which indicates any errors that occurred in the call to **eciSaveDict**.

See Also

eciLoadDict, Synthesis State Parameters

eciSetDefaultParam

Sets the default value for an environment speech parameter.

Syntax

```
int eciSetDefaultParam(
  ECIParam Param
  int iValue
);
```

Parameters

Param
: Parameter value taken from the existing ECIParam enumeration in eci.h. These are the same enumeration values that are used by eciSetParam and eciGetParam.

iValue
: Default value which you want to set for Param.

Return Values

>=0
: The previous default Param value.

-1
: An error. Param or iValue is out of range.

Remarks

Because the initial environment parameter defaults may not be suitable for all operating systems and platforms, this function provides a means for modifying the initial environment parameter defaults to be used on all subsequent speech synthesis engine instances created with eciNew, eciNewEx, eciSpeakText, or eciSpeakTextEx. However, if you already have an existing speech synthesis engine instance and you call eciSetDefaultParam, you must then call eciReset to have these defaults take effect in that instance.

The following table shows the behavior for specific default parameters.

Default Parameter	Behavior
eciLanguageDialect	When called, eciNew, eciSpeakText, or eciReset attempts to load the desired default language. If the desired default language is not available, NULL_ECI_HAND is returned.
eciSampleRate	If a default sample rate selected is not supported by the audio device (determined when eciNew, eciNewEx, eciSpeakText, eciSpeakTextEx, or eciReset is called), then ECI attempts to select another available sample rate for that particular speech synthesis engine instance. The default remains unchanged.

Note

It is the application developer's responsibility to select defaults that are compatible with the functions they are intending to use because changing defaults can affect the behavior of other functions.

See Also

N/A

eciSetDict

Activates a dictionary set for a given ECI instance in the currently active language.

Syntax

```
ECIDictError eciSetDict(
  ECIHand hEngine,
  ECIDictHand dictHandle
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

dictHandle
Handle to the dictionary set.

Return Values

ECIDictError
One of the values enumerated in type ECIDictError. See ECIDictError in Data Types for this enumeration.

Remarks

A dictionary set consists of a Main, Roots, and Abbreviations Dictionary. Each language can be associated with a different dictionary set. **eciSetDict** activates a dictionary set only in that currently active language. The user can call **eciSetDict** multiple times, switching languages before each call.

Dictionary lookups can be deactivated with **eciSetDict** by passing NULL_DICT_HAND to *dictHandle*.

In this case, dictionary lookup is deactivated in all languages simultaneously. Note that this type of dictionary deactivation differs from setting the **eciDictionary** parameter to 1 with a call to

eciSetParam. The latter deactivates lookups in both the internal and user abbreviations dictionaries, but not in any other user dictionaries, and remains in effect when another language becomes active.

See Also

eciNewDict, eciLoadDict, eciGetIndex, User Dictionaries

eciSetOutputBuffer

Sets an output buffer as the synthesis destination.

Syntax

```
Boolean eciSetOutputBuffer(
  ECIHand hEngine,
  int size,
  short* buffer
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

size
: Size of the buffer (in samples). Use 0 to cancel waveform buffer callbacks.

buffer
: Pointer to the buffer to receive PCM audio samples. Use NULL to cancel waveform buffer callbacks.

Return Values

true
: Successfully set the output buffer.

false
: Failure. Does not register a callback. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

Registers your output buffer to receive 16-bit signed PCM audio samples. Signed PCM samples are centered on 0. Calling this function with a size of 0 or a NULL buffer pointer reverts to the default destination and cancels waveform buffer callbacks. Otherwise, when the buffer is full or speech is finished, the **eciWaveformBuffer** message will be sent to your callback.

If a callback has not been registered, **eciSetOutputBuffer** returns false. When an output buffer is successfully registered, device and file output are cancelled. This function may not be called during synthesis.

See Also

eciRegisterCallback, eciSetOutputDevice, eciSetOutputFilename, eciErrorMessage, eciProgStatus

eciSetOutputDevice

Sets an audio output hardware device as the synthesis destination.

Syntax

```
Boolean eciSetOutputDevice(
  ECIHand hEngine,
  int deviceNum
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

deviceNum
: Hardware device number. Values are 0 and higher, or -1.

Return Values

true
: The synthesis destination was successfully set.

false
: An error occurred that prevents output from being sent to this audio device. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

Sets the specified audio device as the destination for synthesis. On Windows, the number of devices is returned by the **waveOutGetNumDevs** call, and devices are numbered sequentially starting with 0. If **deviceNum** is -1, the default destination (device 0) is used. When a device is successfully set, buffer and file output are cancelled. This function may not be called during synthesis.

See Also

eciSetOutputBuffer, eciSetOutputFilename, eciErrorMessage, eciProgStatus

eciSetOutputFilename

Sets a file as the synthesis destination.

Syntax

```
Boolean eciSetOutputFilename(
  ECIHand hEngine,
  const void* filename
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

filename
: Pointer to the name of the file in which to store audio samples. This is a null-terminated C string, which may include a path.

Return Values

true
: The output file was successfully set.

false
: An error occurred that prevented setting the output file. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

Sets the output file as the synthesis destination. If *filename* is NULL, or the C string it points to is empty, the synthesis engine reverts to the default destination. The extension of the filename determines the format of the audio samples written to the file:.

.WAV:	PCM Sound in Windows format
.AU:	u-law with header
.RAU:	raw u-law without header

The three audio formats are not supported on all operating systems. Windows only supports .wav formats.

If the file does not exist, it is created. If the file already exists, output is appended to the existing contents. ECI should only be used in append mode if it is appending to a file that was created by ECI, since it does not currently support all features of complex ".wav" file formats.

When a file is successfully set, buffer and device output are cancelled.

See Also

eciSetOutputBuffer, eciSetOutputDevice, eciErrorMessage, eciProgStatus

eciSetParam

Sets an environment parameter.

Syntax

```
int eciSetParam(
  ECIHand hEngine,
  ECIParam eciParameter,
  int value
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

eciParameter
: One of the values enumerated in type **ECIParam**. See Data Types for this enumeration.

value
: The value the parameter is to be set to, usually 0 or 1. See Synthesis State Parameters for details.

Return Values

>=0
: Success. Returns the previous value.

-1
: Failure. Parameter may be out of range.

Remarks

Sets an environment parameter. Does not affect text already in the input buffer. Returns the previous value on success, or -1 on failure.

See Also

eciAddText, eciGetParam, Voice Parameters

eciSetVoiceName

Sets the name of a voice.

Syntax

```
int eciSetVoiceName(
  ECIHand hEngine,
  int iVoice,
  const void *pBuffer
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

iVoice
: Number of the voice whose parameter value you want to set. Valid values are 0 – 16.

pBuffer
: Pointer to a buffer containing the voice name.

Return Values

true
: The voice name was successfully copied to pBuffer.

false
: An error occurred. Check for valid or out-of-range parameter.

Remarks

The voice name should be in the same code page currently selected for this speech synthesis engine instance. If a Unicode code page is active, *pBuffer* should be in wide-character (Unicode) format with a 16-bit terminator. Otherwise, *pBuffer* should be terminated with an 8-bit, NULL character.

Your buffer should be less than or equal to ECI_VOICE_NAME_LENGTH characters long. Otherwise, it will be truncated. The names of voices 1 - 8 cannot be set; they are read only.

See Also

eciGetVoiceName

eciSetVoiceParam

Sets a voice parameter.

Syntax

```
int eciSetVoiceParam(
  ECIHand hEngine,
  int voiceNum,
  ECIVoiceParam voiceParameter,
  int value
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

voiceNum
: Number of the voice whose parameter value you want to set: 0, 9-16.

voiceParameter
: Voice parameter whose value you want to set:

  eciGender
  eciHeadSize
  eciPitchBaseline
  eciPitchFluctuation
  eciRoughness
  eciBreathiness
  eciSpeed
  eciVolume

value
: Value to set for *voiceParameter*. See Voice Parameters for a description of values, ranges, and defaults.

Return Values

>=0
: Success. Returns the previous value.

-1
: Failure. Parameter may be out of range.

Remarks

Sets the specified voice parameter value for the specified voice. A voice of 0 indicates the active voice. See Voice Parameters for more information.

Returns the previous parameter value, or -1 for error. On error, check the voice number, parameter number, and value.

See Also

eciGetParam, Voice Parameters

eciSpeaking

Indicates whether synthesis is in progress.

Syntax

```
Boolean eciSpeaking(
  ECIHand hEngine
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

true
: Synthesis is in progress.

false
: Synthesis is not in progress.

Remarks

Use this function to poll the ECI instance for synthesis in progress. This function provides an alternative to blocking your thread's execution on **eciSynchronize** during synthesis.

When your application calls **eciSpeaking**, it gives the ECI instance an opportunity to check for messages from the synthesis engine. If you have registered a callback, and there are one or more callback messages from the engine, your callback will be executed on your thread from within this function.

If accurate timing and synchronization are issues for your application, then you should call this function frequently enough that messages and callbacks from the engine can be serviced in a timely manner.

See Also

eciRegisterCallback, eciSynchronize

eciSpeakText

Synthesizes text to the default audio device.

Syntax

```
Boolean eciSpeakText(
  ECIInputText szTextPhrase,
  Boolean fAnnotations
);
```

Parameters

szTextPhrase
Text to be spoken. This is a null-terminated C-string.

fAnnotations
Boolean value that indicates whether annotations are embedded in *szTextPhrase*:

True

There are annotations in *szTextPhrase*. Annotations are to be interpreted.

False

There are no annotations in *szTextPhrase*. Any annotations which nevertheless occur in *szTextPhrase* will not be interpreted.

Return Values

true
The requested string was successfully spoken.

false
An error occurred and the requested string was not spoken.

Remarks

Creates a new ECI instance, speaks the null-terminated C string *szTextPhrase* to the default output device, and destroys the ECI instance it created. When *fAnnotations* is true, annotations within the text are processed, and the speech output is altered accordingly.

Does not return until all speech has been synthesized and played.

The function **eciSpeakText** always speaks the default language unless language-annotated text is passed to it. To change the language, you must insert a language annotation into the text. For example, if you have English and French installed, and *szTextPhrase* contains text in French, you can insert an annotation to switch to the French synthesis engine. See the example below for details on language annotation.

If you set *fAnnotations* to false, but there are annotations in *szTextPhrase*, then the synthesis engine reads them as normal text.

Example

```
#include <stdio.h>
#include "eci.h"

void showMessage( char *msg )
{
  printf( msg );
  getchar();
}

int main( int argc, char *argv[] )
{
  ECIInputText szTextPhrase = "Hello world. `l3.0 Bonjour le monde."
  if ( eciSpeakText(szTextPhrase, true) )
    showMessage( "Success!\n" );
  else showMessage( "Failed.\n" );
  return 0;
}
```

See Also

eciAddText, eciNew, eciSynchronize, eciDeactivateFilter

eciSpeakTextEx

Synthesizes text to the default audio device with the indicated language, dialect and character set.

Syntax

```
Boolean eciSpeakTextEx(
  ECIInputText text,
  Boolean bAnnotationsInTextPhrase,
  ECILanguageDialect value
);
```

Parameters

text
Text to be spoken. This is a null-terminated C-string

bannotationsInTextPhrase
Boolean value that indicate whether annotations are embedded in *text*:

True indicates annotations are in *text* and are to be interpreted.
False indicates annotations are not in *text*.

value
Value to indicate the ECI language dialect.

Return Values

true
The requested string was successfully spoken.

false
An error occurred and the requested string was not spoken.

Remarks

Creates a new ECI instance, speaks the null-terminated C-string *text* to the default output device, and destroys the ECI instance it created. When *bAnnotationsInTextPhrase* is true, annotations within the text are processed, and the voice and speaking characteristics are changed as appropriate.

The function **eciSpeakTextEx** speaks in the indicated language and dialect.

You may enter a language-specific annotation to change the indicated language, dialect, and character set. For example, if you have English and French installed, and the text in *text* is in French, you can insert an annotation indicating that the French TTS engine should be used. See the example.

If you set *bAnnotationsInTextPhrase* to false, but there are annotations in *text*, then the synthesis engine attempts to read them as normal text.

Example

```
#include <stdio.h>
#include "eci.h"

void showMessage( char *msg )
{
  printf( msg );
  getchar();
}

int main( int argc, char *argv[] )
{
  ECIInputText text = "Hello world. `l3.0 Bonjour le monde."
  if ( eciSpeakTextEz(text, true, eciGeneralAmericanEnglish) )
    showMessage( "Success!\n" );
  else showMessage( "Failed.\n" );
  return 0;
}
```

See Also

eciAddText, eciNew, eciSynchronize, eciDeactivateFilter

eciStop

Stops synthesis.

Syntax

```
Boolean eciStop(
  ECIHand hEngine
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

true
: Synthesis is successfully stopped, and the input and output buffers are cleared.

false
: An error occurred that prevented the termination of synthesis. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

Aborts any synthesis in progress, clears the input buffer, clears the output buffer, and releases the audio device if it has been claimed. This function is synchronous, so ECI will have stopped processing before **eciStop** returns.

If the active voice has been changed by annotations during synthesis, then the state of the active voice is undefined when **eciStop** returns. The active voice should be reset by your application to an appropriate setting.

See Also

eciPause, eciSynthesize, eciSpeaking, eciErrorMessage, eciProgStatus

eciSynchronize

Waits in an efficient state until all synthesis is finished.

Syntax

```
Boolean eciSynchronize(
  ECIHand hEngine
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

true
Synthesis has successfully finished.

false
An error occurred or synthesis is not taking place. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

If the synthesis destination is a device, this function does not return until the audio has been played on the device.

While waiting, the ECI instance calls your callback, if you have registered one, and if the synthesis engine has produced any messages for your callback.

Alternatives to **eciSynchronize** are:

- Insert an index using **eciInsertIndex** and wait in a message loop until your callback function is called with that index.
- Wait in a loop that polls **eciSpeaking** and processes your application messages.

See Also

eciSpeaking, eciInsertIndex, eciErrorMessage, eciProgStatus

eciSynthesize

Starts synthesis of text in an input buffer.

Syntax

```
Boolean eciSynthesize(
  ECIHand hEngine
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

true
Synthesis has started.

false
An error occurred that prevented synthesis from starting. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

Starts synthesis of all text in the input buffer. Returns immediately.

It is important to call this function after the last text of an utterance has been passed to **eciAddText** so that the synthesis process will begin, even if **ECISynthMode** is **sentence**, so that sentence ends may be determined correctly.

For example, if you make one call to **eciAddText** with *text* set to "The value of pi is 3.", IBM TTS must wait for the next call to **eciAddText** to see if the sentence ends there, in case the next addition to the text buffer begins "14159". Calling **eciSynthesize** after all text buffers have been sent tells IBM TTS that the last sentence is complete and can be synthesized.

To synthesize text in line-oriented format, such as a table or list, call **eciAddText** and **eciSynthesize** for each line, to ensure that each line is spoken as a separate sentence.

See Also

eciAddText, eciSynchronize, eciErrorMessage, eciProgStatus

eciSynthesizeFile

Synthesizes the contents of a file.

Syntax

```
Boolean eciSynthesizeFile(
  ECIHand eciHandle,
  const void* filename
);
```

Parameters

eciHandle
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

filename
: Pointer to the name of the text file (which may include a path) whose contents you want to synthesize. This is a null-terminated C string.

Return Values

true
: The file was successfully opened.

false
: Error opening file. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information. See remarks.

Remarks

eciSynthesizeFile returns immediately, and does not wait for synthesis to complete.

Opens the named file and starts reading its contents. If the file does not exist, returns false with no other error flags set. This function is equivalent to sending all the text in the named file using **eciAddText,** followed by a call to **eciSynthesize.**

Your application should wait for synthesis to complete before terminating. Use **eciSpeaking** or **eciSynchronize** for this purpose.

See Also

eciAddText, eciSpeaking, eciSynchronize, eciErrorMessage, eciProgStatus

eciTestPhrase

Synthesizes a test phrase.

Syntax

```
Boolean eciTestPhrase(
  ECIHand hEngine
);
```

Parameters

hEngine
: Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

Return Values

true
: Success

false
: Failure. Check **eciErrorMessage** and/or **eciProgStatus** for additional error information.

Remarks

Aborts synthesis if it is underway, clears the input and output buffers, sets the active voice to preset voice 1, loads the sentence “1 2 3.” starts synthesis, and returns.

Your application should wait for synthesis to complete before terminating. Use **eciSpeaking** or **eciSynchronize** for this purpose.

See Also

eciSpeaking, eciSynchronize, eciErrorMessage, eciProgStatus

eciUpdateDict

Updates a dictionary volume with a key/translation pair.

Syntax

```
ECIDictError eciUpdateDict(
  ECIHand hEngine,
  ECIDictHand dictHandle,
  ECIDictVolume whichDictionary,
  ECIInputText pkey,
  ECIInputText ptranslationValue
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

dictHandle
Handle to the dictionary set. This the value returned by **eciNewDict** or **eciGetDict**.

whichDictionary
Enumeration value representing the particular dictionary volume:

eciMainDict – main dictionary
eciRootDict – root dictionary
eciAbbvDict – abbreviation dictionary

pkey
Pointer to a key to the entry in this dictionary. The key is a constant C string.

ptranslationValue
Pointer to a translation value for this entry in this dictionary. The translation value is a constant C string.

Return Values

ECIDictError

One of the values enumerated in type ECIDictError. See ECIDictError in Data Types for this enumeration.

Remarks

Updates the dictionary volume. The update action depends on the existence of the *key* and value of the *translationValue* parameter:

- The entry is added when the *key* does not exist and the *translationValue* is not NULL.
- The entry is updated if the *key* already exists and the *translationValue* is not NULL.
- The entry is deleted if the *translationValue* is NULL. An **ECIDictError** is returned which indicates any errors that occurred in the call.

See Also

eciDictLookup, eciGetDict, eciNewDict

eciUpdateDictA

Updates a dictionary volume with a key/translation pair. This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages.

Syntax

```
ECIDictError eciUpdateDictA(
  ECIHand hEngine,
  ECIDictHand dictHandle,
  ECIDictVolume whichDictionary,
  ECIInputText pkey,
  ECIInputText ptranslationValue
  ECIPartOfSpeech PartOfSpeech
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

dictHandle
Handle to the dictionary set. This the value returned by **eciNewDict** or **eciGetDict**.

whichDictionary
Enumeration value representing the particular dictionary volume:

eciMainDict – main dictionary

eciRootDict – root dictionary

eciAbbvDict – abbreviation dictionary eci

eciMainDictExt - main dictionary used only with Asian languages

pKey
Pointer to a key to the entry in this dictionary. This is a constant buffer containing the key.

pTranslationValue
Pointer to a translation for this entry in this dictionary. This is a constant buffer containing the value.

PartOfSpeech

ECIPartOfSpeech enumeration, which specifies the grammatical category.

Return Values

ECIDictError

One of the values enumerated in type ECIDictError. See ECIDictError in Data Types for this enumeration.

Remarks

This function supports all dictionary volumes, including Main Extension Dictionary (eciMainDictExt), which is used for Asian languages. The update action depends on the existence of *pKey* and value of *pTranslationValue*, as follows:

- The entry is added when key does not exist and *pTranslationValue* is non-NULL.
- The entry is updated if key already exists and *pTranslationValue* is non-NULL.
- The entry is deleted if *pTranslationValue* is NULL. An ECIDictError is returned, which indicates any errors that occurred in the call.

The buffer contents should be in the same code page currently selected for this speech synthesis engine instance. If a Unicode code page is active, *pKey* and *pTranslationValue* should be in wide-character (Unicode) format with a 16-bit terminator. Otherwise, *pKey* and *pTranslationValue* should be an 8-bit, NULL-terminated C string.

See Also

eciDeleteDict, eciDictFindFirstA,eciDictFindNextA, eciDictLookupA , eciLoadDict, eciSaveDict, eciSetDict, eciUpdateDictA

eciUpdateFilter

Allow runtime update of the token replacement applied to known fields.

Syntax

```
ECIFilterError eciUpdateFilter(
  ECIHand hEngine,
  ECIFilterHand whichFilterHand,
  ECIInputText key,
  ECIInputText translation);
);
```

Parameters

hEngine
Handle to the speech synthesis engine instance. This is the value returned by eciNew or eciNewEx.

whichFilterHand
Handle to the filter to be updated. This is the value returned by eciNewFilter.

key
Field that requires special handling.

translation
Text that will replace the key.

Return Values

ECIFilterError
One of the values enumerated in type **ECIFilterError**. See Data Types for this enumeration.

Remarks

See Also

eciNewFilter, eciNew, eciNewEx, Custom Filters

eciVersion

Returns the current IBM Text-to-Speech version number.

Syntax

```
void eciVersion(
  char* buffer
);
```

Parameters

buffer
Pointer to the buffer containing the version string. Must be at least 20 bytes. Should not be NULL.

Return Values

None.

Remarks

This function copies the IBM Text-to-Speech version number as a null-terminated C string to the specified buffer, which must have room for at least 20 characters, including the terminating null.

Annotations

Annotations are special codes placed in text to customize the speech output generated by IBM TTS. You can use them for:

- Selecting a Language and Dialect
- Selecting a Voice or Voice Characteristics
- Selecting a Speaking Style
- Modifying Word Emphasis and Tone
- Modifying Phrase-Final Intonation
- Adding Pauses
- Specifying Alternative Pronunciations
- Filters

All annotations must be preceded by at least one unit of white space.

ECI Annotations

An annotation consists of a backquote (`) followed immediately by a string of characters. For example:

`vs5	Use a speaking rate of 5.
`4	Put very heavy emphasis (level 4) on the following word.
`ts2	Pronounce all characters individually by name.

The Symbolic Phonetic Representations provides a complete listing of the available annotations.

Selecting a Language and Dialect

In order to use an annotation for a language or dialect, you must have installed the text-to-speech engine for that language and dialect. For example, you must install the Standard German text-to-speech engine in order for the German annotation to work.

- If you hear the text being pronounced using the accent of the last active language, then the language of the text has not been installed.
- If you get an error message, then the language you want to hear has probably been installed incorrectly.

Selecting a language will change only the language, not the voice characteristics. The last selected voice characteristics will remain in effect.

The ECI language annotation begins with `l (backquote L), followed by a decimal number specifying the language and dialect. If you specify a language/dialect combination that does not exist, (e.g., ‘l2.3), then the annotation will be ignored because there is no dialect of Spanish corresponding to 2.3.

The following table shows the ECI language annotations:

ECI Annotations	Language or Dialect
`l1	English
`l1.0	American English (default)
`l1.1	British English
`l2	Spanish
`l2.0	Castilian Spanish (default)
`l2.1	Mexican Spanish
`l3	French
`l3.0	Standard French (default)
`l3.1	Canadian French
`l4	German
`l4.0	Standard German (default)
`l5	Italian
`l5.0	Standard Italian (default)

ECI Annotations	Language or Dialect
`l6	Chinese
`l6.0	Standard Mandarin Chinese (default) with GBK support
`l6.0.1	Standard Mandarin Chinese with only Pin Yin support
`l6.0.8	Standard Mandarin Chinese with UCS support
`l6.1	Taiwanese Mandarin Chinese with Big5 support
`l6.1.1	Taiwanese Mandarin Chinese with Zhu Yin support
`l6.1.2	Taiwanese Mandarin Chinese with only Pin Yin support
`l6.1.8	Taiwanese Mandarin Chinese with UCS support
`l7	Portuguese
`l7.0	Brazilian Portuguese (default)
`l8	Japanese
`l8.0	Standard Japanese (default) with Shift-JIS support
`l8.0.8	Standard Japanese with UCS support
`19	Finnish
`19.0	Standard Finnish
`113	Norwegian
`113.0	Standard Norwegian (default)
`114	Swedish
`114.0	Standard Swedish (default)
`115	Danish
`115.0	Standard Danish (default)

Choosing a Speaker

ECI annotations for each language and speaker, as indicated.

ECI Annotation	Description
`l1.0 `v1	Set voice to the “Reed,” the American English adult male voice.
`l1.0 `v2	Set voice to “Shelley,” the American English adult female voice.
`l1.0 `v3	Set voice to “Sandy,” the American English child voice.
`l1.0 `v7	Set voice to “Grandma,” the American English elderly female voice.
`l1.0 `v8	Set voice to “Grandpa,” the American English elderly male voice.
`l1.1 `v1	Set voice to the “Justin,” the British English adult male voice.
`l1.1 `v2	Set voice to “Jane,” the British English adult female voice.
`l1.1 `v3	Set voice to “Nicky,” the British English child voice.
`l1.1 `v7	Set voice to “Nanny,” the British English elderly female voice.
`l1.1 `v8	Set voice to “Gramps,” the British English elderly male voice.
`l2.0 `v1	Set voice to the “Carlos,” the Castilian Spanish adult male voice.
`l2.0 `v2	Set voice to “Pilar,” the Castilian Spanish adult female voice.
`l2.0 `v3	Set voice to “Pepe,” the Castilian Spanish child voice.
`l2.0 `v7	Set voice to “Abuela,” the Castilian Spanish elderly female voice.
`l2.0 `v8	Set voice to “Abuelo,” the Castilian Spanish elderly male voice.
`l3.0 `v1	Set voice to the “Jacques,” the Standard French adult male voice.
`l3.0 `v2	Set voice to “Jacqueline,” the Standard French adult female voice.
`l3.0 `v3	Set voice to “Marius,” the Standard French child voice.
`l3.0 `v7	Set voice to “Mamie,” the Standard French elderly female voice.
`l3.0 `v8	Set voice to “Grandpère,” the Standard French elderly male voice.
`l4.0 `v1	Set voice to the “Max,” the Standard German adult male voice.
`l4.0 `v2	Set voice to “Gisela,” the Standard German adult female voice.
`l4.0 `v3	Set voice to “Matti,” the Standard German child voice.
`l4.0 `v7	Set voice to “Oma,” the Standard German elderly female voice.
`l4.0 `v8	Set voice to “Opa,” the Standard German elderly male voice.

ECI Annotation	Description
`l5.0 `v1	Set voice to the “Enrico,” the Standard Italian adult male voice.
`l5.0 `v2	Set voice to “Lucia,” the Standard Italian adult female voice.
`l5.0 `v3	Set voice to “Chicco,” the Standard Italian child voice.
`l5.0 `v7	Set voice to “Nonna,” the Standard Italian elderly female voice.
`l5.0 `v8	Set voice to “Nonno,” the Standard Italian elderly male voice.
`l6.0 `v1	Set voice to the “Li3 Jing4,” the Standard Chinese adult male voice.
`l6.0 `v2	Set voice to “Wang2 Yan4,” the Standard Chinese adult female voice.
`l6.0 `v3	Set voice to “Li3 Dong1 Dong1,” the Standard Chinese child voice.
`l6.0 `v7	Set voice to “Nai3 Nai,” the Standard Chinese elderly female voice.
`l6.0 `v8	Set voice to “Ye2 Ye,” the Standard Chinese elderly male voice.
`l6.1 `v1	Set voice to the “Zhi4 Ming2,” the Taiwanese Mandarin Chinese adult male voice.
`l6.1 `v2	Set voice to “Chun1 Jiao1,” the Taiwanese Mandarin Chinese adult female voice.
`l6.1 `v3	Set voice to “Xiao3 Bu4 Dian3,” the Taiwanese Mandarin Chinese child voice.
`l6.1 `v7	Set voice to “A3 Ma4,” the Taiwanese Mandarin Chinese elderly female voice.
`l6.1 `v8	Set voice to “A3 Gong1,” the Taiwanese Mandarin Chinese elderly male voice.
`l7.0 `v1	Set voice to the “João,” the Brazilian Portuguese adult male voice.
`l7.0 `v2	Set voice to “Cláudia,” the Brazilian Portuguese adult female voice.
`l7.0 `v3	Set voice to “Chico,” the Brazilian Portuguese child voice.
`l7.0 `v7	Set voice to “Avó,” the Brazilian Portuguese elderly female voice.
`l7.0 `v8	Set voice to “Avô,” the Brazilian Portuguese elderly male voice.
`l8.0 `v1	Set voice to “Taroo,” the Standard Japanese adult male voice.
`l8.0 `v2	Set voice to “Hanako,” the Standard Japanese adult female voice.
`l8.0 `v3	Set voice to “Jiroo,” the Standard Japanese child voice.
`l8.0 `v7	Set voice to “Obaachan”, the Standard Japanese elderly female voice.

ECI Annotation	Description
`l8.0 `v8	Set voice to “Taroo,” the Standard Japanese elderly male voice.
`l9.0 `v1	Set voice to “Antti,” the Standard Finnish adult male voice.
`l9.0 `v2	Set voice to “Tarja,” the Standard Finnish adult female voice.
`l9.0 `v3	Set voice to “Pekka,” the Standard Finnish child voice.
`l9.0 `v7	Set voice to “Isoäiti,” the Standard Finnish elderly female voice.
`l9.0 `v8	Set voice to “Isoisä,” the Standard Finnish elderly male voice.

Selecting a Voice or Voice Characteristics

IBM TTS provides five pre-defined (built-in) voices. Each one has a corresponding voice annotation that can be inserted into the text. In addition, there are a variety of annotations which allow you to directly manipulate individual voice characteristics.

Selecting a Voice in the Current Language

Use the following tags or annotations to choose a built-in voice in the current language:

ECI Annotation	Description
`v1	Set voice to the default male voice 1.
`v2	Set voice to the default female voice 1.
`v3	Set voice to the default child voice 1.
`v4	Set voice to the default male voice 2.
`v5	Set voice to the default male voice 3.
`v6	Set voice to the default female voice 2.
`v7	Set voice to the default older female voice 1.
`v8	Set voice to the default older male voice 1.

The voice annotation will stay in effect until you enter a new voice annotation.

Selecting Voice Characteristics

Individual voices derive their uniqueness from a number of physical factors. In addition, an individual's voice can take on different qualities at different times, depending on such things as mood and circumstance. You can modify these attributes using voice characteristics annotations.

ECI Annotation	Description
`vg0	Set **vocal tract** configuration to male.
`vg1	Set **vocal tract** configuration to female.

ECI Annotation	Description
`vbN	Set **pitch baseline** to N. Annotation range is 0-100.
`vhN	Set **head size** to N. Range is 0 (very small head) to 100 (very large head).
`vrN	Set voice **roughness** (creakiness) to N. Range is 0 (smooth) to 100 (rough).
`vyN	Set **breathiness** to N. Range is 0 (not breathy) to 100 (99 is very breathy; 100 is a whisper).
`vfN	Set **pitch fluctuation** to N. Range is 0 (narrow=monotonic) to 100 (wide).
`vsN	Set **speed** of the utterance to N. Annotation range is 0-250.
`vvN	Set speech **volume** to N. Annotation range is 0-100.
(no annotation)	**Reset** the voice to the original characteristics for the selected speaker.

Voice characteristics tags affect the currently selected voice and remain in effect until a new voice or speaker is specified with a ‘vN annotation, or until the annotation is used again with a different value. Restarting the program resets all of the characteristics to their default values.

Selecting a Speaking Style

Use the following tags or annotations to choose a speaking style:

ECI Annotation	Description
`vy100	Set voice to a whisper.
`vf0	Set voice to a monotone.
varies	Restore voice to the speaking style in use before the \Chr\ command was invoked. The equivalent annotation will vary depending on the original breathiness or pitch fluctuation values for the selected speaker.

Modifying Word Emphasis and Tone

Each word in an utterance is pronounced with a level of emphasis relative to other words in the utterance. You can override the default emphasis patterns by placing an annotation before the word you want to modify.

ECI Annotation	Description
`00	Reduced emphasis
`0	No emphasis
`1	Normal emphasis
`2	Added emphasis
`3	Heavy emphasis
`4	Very heavy emphasis

Emphasis level 1, or *normal emphasis*, is the default level of emphasis for a content word, and emphasis level 00 is the default emphasis for a function word. The last content word in an intonation phrase (the nuclear accent) will receive emphasis level 2, unless you annotate the utterance to change the default pattern.

For example, the default emphasis pattern for the phrase *run through fields of barley* is:

Run	through	fields	of	barley
1	1	1	00	2

Reduced Emphasis

The reduced emphasis annotation can be used to reduce a word to a function word.

No Emphasis

When two words form a single compound word (as in *wet suit* in example (b) below), the second word receives less emphasis than the first. The no-emphasis annotation is used to achieve this effect.

```
(a)  He wore a wet suit to work because his umbrella broke.
(b)  He wore a wet `0 suit while diving.
```

Normal Emphasis

The normal-emphasis annotation can be used to mark a word like *can* (in sentence (b)) as a content word rather than a function word.

```
(a) Eating fat can make you fat.
(b) You're going to need a very fat `1 can to hold all those
    peaches.
```

While this annotation is used to assign normal emphasis to a word that would otherwise receive no emphasis, the word will still receive the nuclear accent in appropriate contexts. You can use the added-emphasis annotation to shift the nuclear accent of the phrase to another word.

Added Emphasis

Typically, the last content word in an intonational phrase receives emphasis level 2 automatically. Sometimes, however, it is more appropriate for this emphasis to fall earlier in the phrase. The added-emphasis annotation can be used to mark words in this way. Note that this causes all subsequent words to be de-emphasized.

```
(a) We demand absolute equality.
(b) We demand `2 absolute equality.
(c) We `2 demand absolute equality.
```

Heavy Emphasis and Very Heavy Emphasis

To give added emphasis to a word, you can increase the emphasis level. Note that setting the emphasis level to 4 also causes all preceding words to be de-emphasized. The higher levels of emphasis are also useful in contradicting a previous statement or expressing incredulity.

```
Your brother has a dog named Spot?
No, my brother has a dog named `3 Fido.

It's not `3 Monday, it's `4 Tuesday.

My aunt has a cat named Fido.
Your aunt has a `3 cat named Fido?
```

Assigning Tones to Words

Use the following annotations to assign tones to words:

Annotation	Description
`al	Low Tone
`ah	High Tone (this is the default for content words)
`af	Falling Tone
`ar	Rising Tone
`as	Scooped Tone
`ad	Downstepped Tone

Modifying Phrase-Final Intonation

Use the following annotations to modify phrase-final intonation:

Annotation	Description
`%	Small pitch rise at the end of the phrase
`%%	Continuation rise at the end of the phrase and low pitch on the nuclear accented word of the phrase.
`%%%	Flat, high pitch at the end of the phrase.
`/	Large pitch fall, as at the end of a paragraph. More perceived finality than at the end of a sentence.

The phrase-final annotations must be immediately followed by the punctuation ending the intonational phrase: either a period, comma, exclamation point, question mark, colon, or semicolon. If the required punctuation is missing, the annotation is ignored.

Adding Pauses

You can use the pause annotation to:

- Modify the pause length created by punctuation symbols.
- Insert pauses between words where there is no punctuation.

Use the following annotations to create a pause.

ECI Annotation	Description
`pN	Create a pause N milliseconds long. The maximum length for a single pause annotation is about 330 seconds.
`epfN	The environmental phrase final (epfN) pause is used to adjust the duration of the final pause at the end of a set of sentences. Default: N = 10 Range: N = 0 to 100 Interpretation: At the end of the final sentence (whether explicit or implied), after calculating final pause value in milliseconds, multiply the value by N/10. The range of the multiplier is thus 0.0 - 10.0. The default factor, 10, produces a multiplier of 1.0 (ie, no change). A factor of 0 produces a multiplier of 0.0, thus producing no pause at all. For example: **`epf0 This is the first sentence. This is a test.** No pause after "test." **`epf5 This is a test.** Pause period is 50% of normal. **`epf10 This is a test.** Pause period is 100% of normal. That is, the default pause length is generated. **`epf20 This is a test.** Pause period is twice normal. **`epf100 This is a test.** Pause period is 10 times normal.

Punctuation Pauses

You can add to the length of a pause associated with a punctuation symbol, or you can replace that pause with a pause of a different length.

- To add to the pause, insert a pause annotation after the punctuation.
  ```
  (a)Avoid the following routes:  Thirteen north, Ninety-six west, and
  Thirty-two south.
  (b)Avoid the following routes:  Thirteen north, `p400 Ninety-six west,
  `p400 and Thirty-two south.
  ```
- To replace the pause, place a pause annotation immediately before the punctuation. There should be no white space between the annotation and the punctuation.
  ```
  (c)I thought I saw Kris.  But I'm not sure about it.
  (d)I thought I saw Kris `p250.  But I'm not sure about it.
  ```
- To replace the pause at the end of a set of sentences, use the epfN annotation. The epfN annotation can be placed anywhere. In the following example, the normal pause of the final sentence is changed to 0 and the normal pause remains between the two sentences. Only the last sentence is affected.
  ```
  (e)`epf0 I thought I saw Kris.  But I'm not sure about it.
  ```

Inserting Pauses

Inserting pauses can be useful for synthesizing the hesitations that occur in natural speech:

```
(a)Amy saw him `p450 well `p450 `3 us `p150 last night.
(b)Take the square root of `p450 no `p450 forget that.  Multiply the total
by .05.
```

Filters

Use the following annotations to activate/deactivate a particular filter.

ECI Annotation	Description
`faN	To activate filter N.
`fdN	To deactivate filter N.

N is optional. If N is ommitted, the default filter will be used. The default filter is 0.

Specifying Alternative Pronunciations

Character Spelling Modes

Use the following annotations to set the character spelling mode:

ECI Annotation	Description
`ts0	No special interpretation (default setting).
`ts1	Pronounce only alphanumeric characters by name.
`ts2	Pronounce all characters individually by name.
`ts3	Pronounce alphabet characters according to the International Radio Alphabet. (This is currently limited to U.S. English only.)

Pronouncing Numbers and Years

Use the following annotations to pronounce numbers and years:

ECI Annotation	Description
`ty0	Pronounce 4-digit numbers as "nonyears."
`ty1	Pronounce 4 digit numbers as "years" (default setting).

Dictionary Processing of Abbreviations

Use the following annotations to process abbreviations:

ECI Annotation	Description
`da0	Turn off both internal and user Abbreviations Dictionary lookups.
`da1	Use the abbreviation dictionaries (default setting).

See User Dictionaries for more information on the user's Main, Roots, and Abbreviations Dictionaries.

Entering Symbolic Phonetic Representations

Use the following annotation to enter SPRs:

Annotation	Description
`[*SPR*]	Pronounce the word phonetically as contained in *`[SPR]*.

Unlike other annotations, the `[*SPR*] annotation does not modify the word or words which follow it. Instead, it is used in place of the word for which it specifies a pronunciation.

See the section titled SPR Tables for more information and tables of symbols in US English and other languages.

Custom Filters

IBM Text-to-Speech offers the ability to transform textual data prior to synthesis through custom filters. These *plug-in* libraries are ideal for transforming complex raw data into more a suitable format for listening. Although dictionaries are good means of transforming text, some transformations may involve more than simple text substitution. Transformation of text through filters in conjunction with an active dictionary is the most powerful combination.

The following example is the conversion of an E-mail message to a format that is much easier to understand.

```
From: root <root@lab.ibm.com>
Full-name: CHRISTINA
Message-ID: <blahblahblah@aol.com>
Date: 10 Oct
Subject: Humans find it hard to listen to all the tagged fields
To: tofield@aol.com
CC: cc@aol.com
MIME-Version: 1.0
Content-Type: multipart/mixed;
boundary="part2_13.cc11a1a.272ef411_boundary"
X-Mailer: Windows AOL sub 125

Email may become complex and embed some strange colloquialisms.

;)
:)
;-)
```

‘Raw’ Email Data (Complex)

```
This message is from:  root.
The message was sent:  10 October.
The subject of this message is:  Humans find it hard to listen to all the
tagged fields.
Email may become complex and embed some strange colloquialisms.
```

‘Filtered’ Email Data (Simplified)

In the example the filter was used to drop lines such as Message-ID:, CC:, and X-Mailer:. The filter was also used to insert ‘This message is from’ and ‘This message was sent’, and to remove the “smiley faces” from the text. A dictionary, however, was used to convert Oct to October. Notice that a dictionary (which simply replaces text) alone cannot remove text based solely on what the line begins with.

Many other transformations (raw stock quotes, directions, recipes, movie listings, etc.) are possible with custom filtering. However the order in which the transformations take place is very important. Currently the following rules apply:

1. An available filter is selected as the active filter.
2. The active filter is then applied to the text.
3. Finally, dictionary rules are applied.

Implementing a Custom Filter

The IBM Text-to-Speech SDK provides a mechanism to develop both “static” and “dynamic” filters. A static filter is one that the client application links with directly at build time. A static filter allows client applications that use a version of the IBM Text-to-Speech runtime prior to version 6.2.2.1 to use text filtering. A client application that uses a static filter must call `eciGetFilteredText` to determine the value of an input string once it has been run through the filter. The value returned in *filteredText* is the string to send to `eciAddText` for synthesis. A dynamic filter, however, is loaded at runtime, so client applications do not need to be aware of it at build time. No separate call to `eciGetFilteredText` is required to cause filtering. If a filter has been created and activated, the active filter will automatically filter text that is sent to the synthesis engine with eciAddText. Additionally, using dynamic filters allows client applications to use more than one type of filter. For client applications that will use the IBM Text-to-Speech version 6.2.2.1 or later, it is recommended that they use dynamic filtering.

Dynamic Filters

Implementing a dynamic filter consists of creating a library that implements the following methods of the Filter base class:

- **`virtual ECIFilterError filterText(const char *input, char **filteredText, bool forceFiltering = false);`**

This is the method that performs the text filtering. It is called from `eciGetFilteredText` and `eciAddText` if the filter is active. The filter object is responsible for the allocation and deallocation of the *filteredText* buffer The value of *filteredText* is only valid until the next call to `filterText` or `deleteFilter`. If *forceFiltering* is true, the filter object will filter the input text even if it is not active (for example, if the client application calls `eciGetFilteredText` on an inactive filter).

- **`virtual ECIFilterError activateFilter();`**

This method is what sets the filter to the "active" state via `eciActivateFilter`.

- **`virtual ECIFilterError deactivateFilter();`**

This method is what deactivates the active filter via `eciDeactivateFilter`.

- **`virtual ECIFilterError deleteFilter();`**

This method is what is responsible for deleting the filter and performing any required cleanup of the filter object. This method is called via `eciDeleteFilter`.

- **`virtual ECIFilterError updateFilter(const char *key, const char *translation);`**

This is the method that will set the rules used to filter text that is processed with `filterText`. It is called from `eciUpdateFilter`.

- **`FILTER_API Boolean getObject(unsigned long idInterface, void **ppUnknown)`**

This function is the required entry point of the dynamic filter. It is responsible for creating the Filter object and returning a filter handle to the client. It is called from `eciNewFilter`.

Dynamic Filter Sample Code

```
#include “simplefilter.h” // definition of the SimpleFilter class
simpleFilter::simpleFilter() : filteredText(NULL) {
} // simpleFilter::simpleFilter()
simpleFilter::~simpleFilter() {
   if (filteredText) {
      delete [] filteredText;
      filteredText = NULL;
   } // if (filteredText)
} // simpleFilter::~simpleFilter()
ECIFilterError simpleFilter::deleteFilter() {
   delete this;
   return FilterNoError;
} // ECIFilterError simpleFilter::deleteFilter()

ECIFilterError simpleFilter::filterText(const char *input, char
**clientFilteredText, bool forceFiltering) {
if (forceFiltering || isActive()) {
   if (filteredText) {
      delete [] filteredText;
      filteredText = NULL;
   } // if (filteredText)
   // TODO: Insert code to filter text here
   // Filter object should allocate a new filteredText buffer and send
   // a pointer to that buffer back to the client.
} // if (forceFiltering || isActive())
   else {
      // copy input to filteredText
   }
   *clientFilteredText = filteredText.
   return FilterNoError;
} // int simpleFilter::filterText(const char *input, char *filteredText)

} // FILTER_API Boolean getObject(unsigned long idInterface, voice
**ppUnknown)
} // extern "C"
#endif // __cplusplus
```

```
#ifdef __cplusplus
extern "C" {
FILTER_API Boolean getObject(unsigned long idInterface, void
**ppUnknown)
{
   if ((idInterface == ID_UNKNOWN) || (idInterface == ID_FILTERINST)) {
      void *pFilter = NULL;
      pFilter = new simpleFilter();
      if (pFilter) {
         *ppUnknown = pFilter;
      } // if (pFilter)
   } // if ((idInterface == ID_UNKNOWN) || (idInterface ==
ID_FILTERINST))
   return (*ppUnknown != NULL);
```

Dynamic Filter Sample Code

Once the custom dynamic filter is developed, it must be installed for client applications to be able to access it. A filter number identifies dynamic filters for a language/dialect combination. To make a dynamic filter available to client applications, it must be added to the registry on Windows platforms or to the eci.ini file on Unix platforms.

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\ViaVoice Outloud
5.0\ECIINI\1.0\Path_Filter1 = c:\Program Filter\IBM TTS 6.22
SDK\samples\simplefilter\release\simplefilter.dll
```

Registry example (Windows platforms)

```
[1.0]
Path=enu50.syn
Path_Filter1=/usr/lpp/viavoice/sdk/samples/simplefilter/release/
libsimplefilter.a
```

eci.ini file example (Unix platforms):

The Filter base class implementation is contained in the header file `filter.h` and the library `filters.lib` (`libfilters.a` on Unix platforms).

The IBM Text-to-Speech SDK includes a complete example of a simple dynamic filter (SimpleFilter).

NOTE:

Client applications that will use a dynamic filter should not include `ecifilter.h`.

Static Filters

Implementing a static filter consists of creating a library that implements the following functions:

- `eciNewFilter`
- `eciActivateFilter`
- `eciDeleteFilter`
- `eciUpdateFilter`
- `eciGetFilteredText`
- `eciDeactivateFilter`

Static Custom Filter Sample Code

```
#include “ecifilter.h”
#include "eci.h"

ECIFilterHand eciNewFilter(ECIHand hEngine, unsigned int filterNumber) {
// Insert code for eciNewFilter here
} // ECIFilterHand eciNewFilter(ECIHand hEngine, unsigned int
filterNumber)

ECIFilterError eciActivateFilter(ECIHand hEngine, ECIFilterHand
whichFilterHand) {
// Insert code to eciActivateFilter here
} // ECIFilterError eciActivateFilter(ECIHand hEngine, ECIFilterHand
whichFilterHand)

ECIFilterHand eciDeleteFilter(ECIHand hEngine, ECIFilterHand
whichFilterHand) {
// Insert code for eciDeleteFilter here
} // ECIFilterHand eciDeleteFilter(ECIHand hEngine, ECIFilterHand
whichFilterHand)

ECIFilterError eciUpdateFilter(ECIHand hEngine, ECIFilterHand
whichFilterHand, ECIInputText key, ECIInputText translation) {
// Insert code for eciUpdateFilter here
} // ECIFilterError eciUpdateFilter(ECIHand hEngine, ECIFilterHand
whichFilterHand, ECIInputText key, ECIInputText translation

ECIFilterError eciDeactivateFilter(ECIHand hEngine, ECIFilterHand
whichFilterHand) {
// Insert code for eciDeactivateFilter here
} // ECIFilterError eciDeactivateFilter(ECIHand hEngine, ECIFilterHand
whichFilterHand)

ECIFilterError eciGetFilteredText(ECIHand hEngine, ECIFilterHand
whichFilterHand, ECIInputText input, ECIInputText *filteredText) {
// Insert code for eciGetFilteredText here
} // ECIFilterError eciGetFilteredText(ECIHand hEngine, ECIFilterHand
whichFilterHand, ECIInputText input, ECIInputText *filteredText)
```

Static Custom Filter Sample Code

A client application that wishes to use the static filter must include `ecifilter.h` before including `eci.h` and link with the filter library.

The IBM Text-to-Speech SDK version 6.2.2.1 includes a static email filter. To use the static email filter, a client application is compiled with `ecifilter.h` and linked with `mailfilter.lib` (`libmailfilter.a` on Unix platforms).

Sample client code using the default dynamic filter:

```
#include "eci.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{

ECIFilterHand filterObject;
ECIHand handle;

ECIDictHand dictHand;
ECIDictError drc;

ECIInputText filteredText;

handle = eciNewEx(eciGeneralAmericanEnglish);
eciSetParam(handle, eciSynthMode, 1);
eciSetParam(handle, eciInputType, 1);
eciSetParam(handle, eciSampleRate, 0);

dictHand = eciNewDict(handle);
/* maildict.dct contains the translations for some of the more common
email jargon and
abbreviations */
drc = eciLoadDict(handle, dictHand, eciMainDict, “.\\maildict.dct");
eciSetDict(handle, dictHand);

}
```

```
filterObject = eciNewFilter(handle);
eciActivateFilter(handle, filterObject);

if (argc > 1)
{

FILE *fp = fopen(argv[1], "r");
char *charBuffer = new char[256];
void *ptr = (void *) charBuffer;
if (fp){
while ((ptr = (void *) fgets((char *) ptr, 256, fp)) != NULL) {

eciAddText(handle, ptr); /* The filtered text is synthesized. */
}
}
eciSynthesize(handle);
eciSynchronize(handle);
fclose(fp);
getchar();
}
eciDeleteFilter(handle, filterObject);
eciDeleteDict(handle, dictHand);
eciDelete(handle);
return 0;
```

Sample client code using the default dynamic filter

Symbolic Phonetic Representations

A Symbolic Phonetic Representation (SPR) is the phonetic spelling used by IBM TTS to represent the pronunciation of a single word. An SPR represents the sounds of the word, how these sounds are divided into syllables, and which syllables receive stress.

SPRs can be generated by IBM TTS as output (see eciGeneratePhonemes), and they can be used as input into IBM TTS in order to specify pronunciations which are not produced by the ordinary letter-to-sound rules.

You can enter an SPR directly into a text in place of the ordinary spelling of a word, or you can enter an SPR as the translation value of either a main dictionary or roots dictionary entry, so that the desired pronunciation is generated whenever that word is encountered in any text (see Main Dictionary (eciMainDict) and Roots Dictionary (eciRootDict) for more information on these topics).

SPR Form

An SPR consists of a sequence of allowable SPR symbols for a given language, enclosed in square brackets [] and preceded by a backquote character `. For example, the following are valid SPRs in English:

```
though   `[.1Do]
shocking `[.1Sa.0kIG]
```

A period signals the beginning of a new syllable, the digits 1 and 0 indicate the stress level of the syllables, and the letters D, o, S, a, k, I, and G represent specific English speech sounds. Each of these elements of the SPR is discussed in further detail below. An SPR entry which does not follow the requirements detailed below is invalid, and is spelled out character by character.

Syllable Boundaries

A period is used to mark the beginning of each syllable in the SPR output generated by IBM TTS. However, periods are optional in SPR input in all languages, and, except in German, will not have any effect on the way the word is syllabified by the text-to-speech rules. In German, a period can be used in SPR input to trigger a syllable boundary at the specified location (see German SPRs).

Syllable Stress

Syllables can be marked for stress with the digits 1, or 2, or 0, for primary stress, secondary stress, and no stress, respectively. Some languages do not use secondary stress and thus do not accept the use of the digit 2 in SPRs; see sections on specific languages below. If a word has more than one syllable, at least one of these syllables must be marked for primary stress, or the SPR will be considered invalid and will be read out character by character. Other syllables can be marked with either secondary or no stress. Syllables that are not marked for stress are assumed to have no stress.

The syllable stress marker (1, 2, or 0) should be within the syllable's boundaries but always to the left of the vowel of the syllable. If you do not know where the syllable boundaries in a word like construction are located, any of the following SPRs will correctly place the primary stress on the vowel in bold type:

"construction"
`[kXn1str**H**kSXn]

`[kXns1tr**H**kSXn]
`[kXnst1r**H**kSXn]
`[kXnstr1**H**kSXn]

In French, the syllable stress marker must directly precede the vowel of the syllable (see examples in the section on French SPRs).

Speech Sound Symbols

Each language uses its own inventory of SPR symbols for representing the speech sounds of that language. The section SPR Tables contains tables of valid SPR symbols for the sounds of each language, with examples of words in which each sound occurs. Letters are case-sensitive, so ‘[e] and ‘[E] represent two different sounds. Two-character symbols must be contained in single quotes; for example, German *heim* `[h'aj'm]. SPRs containing sound symbols that are not allowable in the current language will be considered invalid, and spelled out character by character.

The sounds of every language have specific distributional patterns within that language. For example, in all dialects of English, the sound [G] of sing `[.1sIG] does not occur at the beginning of a word. Other American English sounds that have a particularly narrow distribution are the glottal stop [?], the flap [F], and the syllabic nasal [N]. (See American English SPRs). If you enter a sound symbol in a context where it does not normally occur, the resulting speech may sound unnatural.

IBM TTS applies a sophisticated set of linguistic rules to its input to reflect the processes by which sounds change in specific contexts in natural language. For example, in American English, the sound [t] of write `[.1r1Yt] is pronounced as a flap [F] in writer `[.1rY.0FR]. SPR input will undergo these modifications just as ordinary input text does. In this example, whether you enter `[.1rY.0tR] or `[.1rY.0FR], the output of the program will be the same.

SPR Tables

The following tables show the inventory of allowable SPR symbols in each IBM TTS language/dialect. Each sound symbol is accompanied by examples showing typical spellings of the sound in actual words, with the letters representing the given sound underlined. (Due to dialectal differences, the examples may not always match your pronunciation.) Remarks specific to SPRs in individual languages are also included in the appropriate sections. Refer to SPR Form for general guidelines on creating and using SPRs.

American English SPRs

Regular Vowels

The following table includes the symbols for regular vowels.

American English Symbol	Example Words
a	father, lot
A	back, had
e	cake, pain
E	hedge, let
i	see, speak, believe
I	pick, ill
o	both, oak
c	law, cough
u	zoo, truth
U	took, put
H	but, mug, son
R	butter, hurt

Diphthongs

The following table includes the symbols for diphthongs.

American English Symbol	Example Words
O	toil, boy
W	out, cow
Y	life, fine

Reduced Vowels

The following table includes the symbols for reduced vowels.

American English Symbol	Example Words
x	sofa, alone, suppose, tedious, America
X	roses, connect, melody, symphony, hinted

Consonants

The following table includes the symbols for consonants.

American English Symbol	Example Words
b	bad, sob
p	pit, rip
d	dip, had
t	tip, pet
g	good, bug
k	kill, cat, make, back
D	this, breathe
T	thing, Beth
v	vase, save
f	field, if, graph
z	zip, phase

American English Symbol	Example Words
s	seal, miss, ceiling
Z	treasure, garage
S	ship, wish
J	Jane, huge
C	chip, witch, nature
h	hot, hero
m	man, hum, summer
n	never, sun, winner
G	sing, finger
r	borrow, rake
l	low, hall
w	wear, quick
y	yes, Virginia
M	hmmm
? ("glottal stop")	kitten, Latin
F ("flap")	writer, fiddle
N ("syllabic nasal")	button, satin, eaten, burden

Syllable Stress

The following table includes the symbols for syllable stress.

1	primary stress (most prominent stress in the word)
2	secondary stress
0	no stress

Syllable Boundary

The following table includes the symbol for a syllable boundary.

.	(period) beginning of a syllable

British English SPRs

Regular Vowels

The following table includes the symbols for regular vowels.

British English Symbol	Example Words
a	path, father, chant
A	back, had
e	cake, pain
E	hedge, let
i	see, speak, believe
I	pick, ill
o	both, oak
c	law, court, hall, water
@	rod, cough
u	zoo, truth
U	took, put
H	but, mug, son
R	butter, hurt

Diphthongs

The following table includes the symbols for diphthongs.

British English Symbol	Example Words
O	toil, boy
W	out, cow
Y	life, fine

Reduced Vowels

The following table includes the symbols for reduced vowels.

British English Symbol	Example Words
x	sofa, alone, suppose, America
X	roses, hinted

Consonants

The following table includes the symbols for consonants.

British English Symbol	Example Words
b	bad, sob
p	pit, rip
d	dip, had
t	tip, pet
g	good, bug
k	kill, make, back
D	this, breathe
T	thing, Beth
v	vase, save
f	field, if, graph
z	zip, phase
s	seal, miss, ceiling
Z	treasure, garage
S	ship, wish
J	Jane, huge
C	chip, witch
h	hot, hero
m	man, hum, summer
n	never, sun, winner

British English Symbol	Example Words
G	si<u>ng</u>, fi<u>n</u>ger
r	bo<u>rr</u>ow, <u>r</u>ake
l	<u>l</u>ow, ha<u>ll</u>
L	candl<u>e</u>
y	<u>y</u>es, Virgini<u>a</u>
w	<u>w</u>ear, q<u>u</u>ick

Syllable Stress

The following table includes the symbols for syllable stress.

1	primary stress (most prominent stress in the word)
2	secondary stress
0	no stress

Syllable Boundary

The following table includes the symbol for a syllable boundary.

.	(period) beginning of a syllable

German SPRs

Vowels

The following table includes the symbols for vowels.

German Symbol	Example Words
i	lieben, Titel, tief
I	bitte, Tisch, Licht
e	geben, Ehre, See
E	treffen, Geld, kämmen
'E:'	Käse, Mädchen, wägen
a	Haar, haben, fahren
A	lassen, matt, Apfel
u	gut, Uhr, Uwe
U	Hund, Fluß, Mutter
o	Ober, ohne, Boot
O	Kopf, Stopp
y	Bücher, Tür, kühn
Y	fünf, füllen, Künstler
'oe'	Löwe, hören, Söhne
'OE'	können, hölzern, östlich
@	bitte, Kamera, Boden

Diphthongs

The following table includes the symbols for diphthongs.

German Symbol	Example Words
'aj'	heim, Waise, Mai

German Symbol	Example Words
'aw'	Haus, Maul, Frau
'oj'	heute, Gebäude, Häuser

Nasalized Vowels (occur in borrowings only)

The following table includes the symbols for nasalized vowels.

German Symbol	Example Words
'a~'	Chance
'E~'	Teint
'o~'	Pardon
'oe~'	Parfum

Consonants

The following table includes the symbols for consonants.

German Symbol	Example Words
b	Boden, Bett, oben
p	Papier, Lippe, Grab
d	dunkel, kindisch, Helden
t	Tag, bitte, Rad
g	geben, grau, Tage
k	Katze, Ecke, Skulptur, lag, quitt
v	Wagen, viskös, Volum, oval
f	fast, hoffen, Vater
z	See, Satz, lesen
s	Fuß, lassen, Last, Haus
Z	Garage, Genie
S	schon, spielen, Stil, wäscht

German Symbol	Example Words
X	ich, Chemie, Kelch, mancher
x	Buch, Bach, Wochen
P	Pflanze, Stumphen
T	Zauber, Polizei, Glanz
J	Job, Dschungel
C	deutsch, Chile, Cello
m	Mann, kommen, Atem
n	Nacht, können, Kind
G	Finger, längs, Anfang
l	lesen, fallen, Pult
r	Rad, führen
R	Wieder, über
j	Junge, ja, Jahr, Ministerium
w	Eduard, aktuell, Januar
h	hoch, Hand, Ahorn

Syllable Stress

The following table includes the symbols for syllable stress.

1	primary stress (most prominent stress in the word)
2	secondary stress
0	no stress

Syllable Boundary

The following table includes the symbol for a syllable boundary.

.	(period) beginning of a syllable

In German, a period in an SPR entry will trigger a syllable boundary at that location.

Canadian French SPRs

Vowels

The following table includes the symbols for vowels.

French Symbol	Example Words
a	pattes, lac, cave
A	char, bois, mâle
e	café, déformer, été
E	faite, dresser
i	film, typique
I	site, plastique, ride
o	taurillon, vaudevilliste
c	paul, note, échalotte
u	roue, où, tour
U	foule, mousse
y	utile, pure, Bruno
Y	autobus, chute
x	litres, marbre (note: le [x] s'efface dans certains contextes.)
'eu'	meuglement
'oe'	cependant, cheval
'a:'	voyage, information
'e:'	steak (anglicismes)
'E:'	père, annuaire, fête
'o:'	paule, beau, tôt, côté
'c:'	loge, encore
'u:'	four, douze
'y:'	dur, buse

French Symbol	Example Words
'eu:'	jeûne, émeute
'oe:'	peur, jeune, déjeuner
'a~'	banc, en, temps
'E~'	fin, plein, faim
'o~'	bon, pont, mon
'oe~'	un, aucun

Consonants

The following table includes the symbols for consonants.

French Symbol	Example Words
b	bébé, balle, robe
p	porte, prêt, guêpe
d	dort, dolmen
t	ton, patte, théâtre
g	guerre, bague, garer
k	kilo, caler, quai
v	laver, wagon, visiter
f	chef, faim, phare
D	duque, dire
T	petit, tuque
z	jaser, réseau, zigzaguer
s	sans, ambition, façon
Z	rage, gîte, jouer
S	cheval, lâche, schéma
J	jeans, jogging
C	gaucho, gaspacho
m	maman, femme, miser

French Symbol	Example Words
n	Anne, ni, maniaque
'nj'	agneau, campagne
'ng'	parking, camping
r	parer, rare, carreau
l	litre, illisible, pâle
j	hiérarchie, paille, yoga
w	oui, moi, voilà
H	lui, nuit, nuée

Syllable Stress

The following table includes the symbols for syllable stress.

1	primary stress (most prominent stress in the word)
2	secondary stress
0	no stress

In French, the stress marker must immediately precede the vowel of the syllable.

Syllable Boundary

The following table includes the symbol for a syllable boundary.

.	(period) beginning of a syllable

Liaison

In French, the underscore can be used following a word-final consonant (but within the right bracket which closes the SPR) to indicate that it is a liaison consonant: that is, it will be pronounced only if the following word begins with a vowel.

For example, a roots dictionary key *petit* with the translation value `[p'oe't1it_] will have the final [t] pronounced in the input string *un petit ami* but not in the input string *un petit chien*. On the other hand, an entry with the translation value `[nEt] will have the final [t] pronounced regardless of context.

The following examples show how to use the symbol for liaison.

_	(underscore character) allow liaison if the following word begins with a vowel. For example:	
	`[p0'oe't1it_]	The [t] will not be pronounced unless the following word begins with a vowel.
	`[nEt]	The [t] will always be pronounced.

French SPRs

Vowels

The following table includes the symbols for vowels.

French Symbol	Example Words
a	pattes, lac, cave
e	café, déformer, été
E	père, annuaire, mer
i	film, type, rythmique
o	paule, tôt, eaux
c	paul, note, échalotte
u	roue, où, août, tour
y	utile, pure, Bruno
'eu'	peu, jeûner, émeute
'oe'	peur, jeune, déjeuner
'a~'	banc, en, temps
'E~'	fin, plein, faim
'o~'	bon, pont, mon
x	litres, marbre

Consonants

The following table includes the symbols for consonants.

French Symbol	Example Words
b	bébé, balle, robe
p	porte, prêt, guêpe
d	dort, dolmen, addition
t	ton, patte, théâtre

French Symbol	Example Words
g	guerre, bague, garer
k	kilo, caler, quai
v	laver, wagon, visiter
f	chef, faim, phare
z	jaser, réseau, zigzaguer
s	sans, ambition, façon
Z	rage, gîte, jouer
S	cheval, lâche, schéma
m	maman, femme, mettre
n	Anne, ni, anonyme
'nj'	agneau, campagne
'ng'	parking, camping
r	parer, rare, carreau
l	litre, illisible, pâle
j	hiérarchie, paille, yéyé
w	oui, moi, voilà
H	lui, nuit, nuée

Syllable Stress

The following table includes the symbols for syllable stress.

1	primary stress (most prominent stress in the word)
2	secondary stress
0	no stress

In French, the stress marker must immediately precede the vowel of the syllable.

Syllable Boundary

The following table includes the symbol for a syllable boundary.

.	(period) beginning of a syllable

Liaison

In French, the underscore can be used following a word-final consonant (but within the right bracket which closes the SPR) to indicate that it is a liaison consonant: that is, it will be pronounced only if the following word begins with a vowel.

For example, a roots dictionary key *petit* with the translation value `[p'oe't1it_] will have the final [t] pronounced in the input string *un petit ami* but not in the input string *un petit chien*. On the other hand, an entry with the translation value `[nEt] will have the final [t] pronounced regardless of context.

The following examples show how to use the symbol for liaison.

_	(underscore character) allow liaison if the following word begins with a vowel. For example:	
	`[p0'oe't1it_]	The [t] will not be pronounced unless the following word begins with a vowel.
	`[nEt]	The [t] will always be pronounced.

Standard Italian SPRs

Vowels

The following table includes the symbols for vowels.

Italian Symbol	Example Words
a	lasagna, allegro
e	nero, duetto
E	ecco, liceo
i	isola, formica
o	padrone, attore
c	costa, mosse
u	luna, ufficio

Consonants

The following table includes the symbols for consonants.

Italian Symbol	Example Words
b	bocca
p	partire
d	data
t	toccare
g	grande
k	casa, vecchio
v	vano, vivere
f	fare
z	paese, sbaglio
s	pesto, stasera
S	scegliere, lasciare

Italian Symbol	Example Words
J	Giovanni, congelare
C	cece, ciao
D	zabaione
T	pizza, zuppa
m	mamma
n	niente
N	gnocchi, lasagna
r	caro
R	terra
l	lento, palma
L	figlio, gli
y	ieri, rasoio
w	nuovo

Syllable Stress

The following table includes the symbols for syllable stress.

1	primary stress (most prominent stress in the word)
0	no stress

Syllable Boundary

The following table includes the symbol for a syllable boundary.

.	(period) beginning of a syllable

Mexican Spanish SPRs

Vowels

The following table includes the symbols for vowels.

Spanish Symbol	Example Words
a	agua
e	este
i	igual
o	oso
u	uve

Consonants

The following table includes the symbols for consonants.

Spanish Symbol	Example Words
b	basta, hubo
p	parte, apagar
d	dar
t	toma, atar
g	goma, haga
k	coger, irak
L	milla, llueve
f	flaco, afuera
z	mismo, desde
s	si, casa
R	ropa, perro
C	coche, chico
B	daba

Spanish Symbol	Example Words
D	cada
G	hagalo
v	vaca
Z	llave
ng	angel
j	jota, gente
m	mano, amor
n	no, mano
N	España
r	arena, pero
l	loco, algo
y	oigo, tiesto
w	fuera, deuda

Syllable Stress

The following table includes the symbols for syllable stress.

1	primary stress (most prominent stress in the word)
0	no stress

Syllable Boundary

The following table includes the symbol for a syllable boundary.

.	(period) beginning of a syllable

Castilian Spanish SPRs

Vowels

The following table includes the symbols for vowels.

Spanish Symbol	Example Words
a	agua
e	este
i	igual
o	oso
u	uve

Consonants

The following table includes the symbols for consonants.

Spanish Symbol	Example Words
b	basta
B	daba, hubo
p	parte, apagar
d	dar
D	nada
t	toma, atar
g	goma, haga
G	hagalo
k	coger, irak
f	flaco, afuera
z	mismo, desde
s	si, casa
R	ropa, perro

Spanish Symbol	Example Words
T	ciudad, manzana
C	coche, chico
j	jota, gente
m	mano, amor
n	no, mano
N	España
r	arena, pero
l	loco, algo
L	llegar, pollo
y	oigo, tiesto
Y	playa, mayor
w	fuera, deuda

Syllable Stress

The following table includes the symbols for syllable stress.

1	primary stress (most prominent stress in the word)
0	no stress

Syllable Boundary

The following table includes the symbol for a syllable boundary.

.	(period) beginning of a syllable

Brazilian Portuguese SPRs

Vowels

The following table includes the symbols for vowels.

Brazilian Portuguese Symbol	Example Words
a	vira, à, álgebra
e	dedo, português
E	és, belo
I	fizesse, bode
o	cor, bolha
c	próximo, porta
u	cura, campo, peru
‘a~’	campo, lã, atlântico
‘e~’	alguém, tem, agüentar
‘I~’	trinca, assim
‘o~’	tom, cônsul, lições
‘u~’	alguns, um

Consonants

The following table includes the symbols for consonants.

Brazilian Portuguese Symbol	Example Words
b	abre, Brasil, bode
p	pluma, primo, pampa
d	dedal, draga
t	topada, ponto, trinca
g	gato, guarda, português

Brazilian Portuguese Symbol	Example Words
k	cama, kilo, queda
v	vila, breve
f	faixa, flauta, abafado
z	zero, caso, cosmos
s	certo, extra, avançar
Z	geral, jarro, bafejo
S	acho, xícara, baixa
J	bode, diz, adjetivo, admirar
C	bote, tive
m	fome, macaco
n	dona, novo
N	inhame, cunha
r	caro, trem, falar
R	carro, rio, guelra
l	leite, cavalo, claro
L	mal, relva
‘ly’	lhe, bagulho
y	falei
Y	calões, cãibra
w	guardo, meu, agüentar
W	capitão

Syllable Stress

The following table includes the symbols for syllable stress.

1	primary stress (most prominent stress in the word)
0	no stress

Finnish SPRs

Vowels

The following table includes the symbols for vowels.

Finnish Symbol	Example Words
a	tavara
’a:’	taas
A	käsi
’A:’	pääsi
e	ele
’e:’	tullee
i	lihas
’i:’	kiitos
o	kotiin
’o:’	koossa
O	öljyta
’O:’	Töölö
u	puhu
’u:’	luussa
y	yksi
’y:’	pyyhkiä
’ao’	Håkan

Glides

The following table includes the symbols for vowel offglides, which follow a vowel to form a diphthong.

Finnish Symbol	Example Words
I	leipä
U	hius
Y	löyden

Consonants

The following table includes the symbols for consonants.

Finnish Symbol	Example Words
b	tabu
p	poika, kauppa
d	tiede
t	tämä, että
g	teknologia
k	rikas, Pekka
v	vaimo
f	profeetta
z	Zagreb
s	suu, pässi
Z	Zhironofski
S	plyysi
J	Juneau
C	Chile
h	huomenta, vaahtoa
m	maa, ymmäarrä
n	nappi, lennättää

Finnish Symbol	Example Words
G	Helsingin
r	raha, varis, Tarja
R	piirros
l	laula, illalle
j	ja, ajatus
w	Washington

Syllable Stress

The following table includes the symbols for syllable stress.

1	primary stress (most prominent stress in the word)
2	secondary stress
0	no stress

Syllable Boundary

The following table includes the symbol for a syllable boundary.

.	(period) beginning of a syllable

Chinese SPRs

The section describes the SPR for Chinese.

Vowels

The following table includes the symbols for vowels.

Symbol	Example Words
a	ba1, ban4, shuang3
e	ke4, jie2, xue2, me0, er0
i	pi4, zi4, shi1, dui4, xie3, ying1
o	bo1, hong1
u	lu4, jun1, dun1, shuo1
ai	pai3, lai2, hai2er0
ao	pao3, lao2
ei	bei1, bei4er0
ou	you1, hou4, pou1, lou4
u:	nuu3
ue:	lue:4, nue:4

Constants

The following table includes the symbols for consonants.

Symbol	Example Words
p	piao1, pin4
b	bo1, biao3

t	ta1, ting2
d	diao4, du2
k	kai4, kuang1
g	geng4, gu1
z	zai1, zuan3
zh	zhen1, zhuang4
c	ci4, cai2
ch	chen2, chuang3
q	qi4, qian2
j	ji1, jiong4
f	fei1, feng2
s	san1, song4
sh	sheng1, shao3
x	xia4, xue2
h	han3, hua2
m	ma4, ming2
n	na2, duan3, lin2
ng	rang4, nong2
r	rou4, er0
l	le4, liang2, lu1
y	ye1, yong4, ying2
w	wang2

Tone

The following table includes the symbols for tones.

	neutral tone -no need to attach symbol for neutral tone
1	tone 1
2	tone 2
3	tone 3
4	tone 4

Chinese Starting Syllable

The following table includes the symbol for a syllable boundary.

+	beginning of Chinese syllables

Syllable Boundary

The following table includes the symbol for a syllable boundary.

.	(period) beginning of a syllable

Chinese SPR example

The following table shows examples of Chinese SPRs.

Word	SPR
Jin1Tian1	`[+.jin1.tian1]
Ni2Hao3	`[+.ni2.hao3]
Zai4Jian4	`[+.zai4.jian4]

Note

Traditional Chinese TTS also uses the same SPR as Simplified Chinese does (not ZhuYin). Simplified Chinese and Traditional Chinese TTS can also accept mixed language (Chinese/ English) SPRs, such as:

```
`[+.ni2.hao3] `[.1tam]
```

Japanese SPRs

This section describes the SPR symbols for Japanese vowels and consonants.

Vowels

The following table shows the symbols for Japanese vowels.

Symbol	Example Words
a	sake, haba
A	maaku, meekaa
e	te, suteru
E	keisatsu, heisei
i	ima, miru
I	ii, atarashii
o	hodo, soba
O	koujou, oozei
u	fuyu, sushi
U	kyuushuu, yuumei

Consonants

The following table shows the symbols for Japanese consonants.

Symbol	Example Words
b	kabi, chiba
p	shimpai, kappa
d	mada, deru
t	toki, atta
g	tamago, jitusgyou

Symbol	Example Words
k	kakaru, sakka
f	gifu, futatsu
z	mizu, zasshi
s	sakana, issei
S	shima, isshou
'ts'	tsunami, tatsu, ittsuu
'dZ'	jibun, meiji
'tS'	chizu, machi, micchaku
h	haba, iroha, hima, zehi, hyouban
m	maru, sama
n	nori, hana
N	shimbun, gengo, insatsu, jin
r	ryokan, kiru
y	yoru, toyota, kyoukai
w	wareware, awa

Examples

The following table shows examples of Japanese SPRs.

Word	SPR
gambaru	`[.0gaN.1ba.0ru]
kesson	`[.1kes.0soN]
obaasan	`[.0o.1bA.0saN]
chiji	`[.1’tS’i.0’dZ’i]
isshoukenmei	`[.0iS.0SO.1keN.0mE]

Word	SPR
ryokakki	`[.0ryo.1kak.0ki]
micchaku	`[.0mit.0’tS’a.0ku]
oosakaben	`[.0O.0sa.0ka.0beN]
insatsubutsu	`[.0iN.0sa.1’ts’u.0bu.0’ts’u]
kudamono	`[.0ku.1da.0mo.0no]
ittsuu	`[.0it.1’ts’U]
atarimae	`[.0a.0ta.0ri.0ma.0e]
hikkosu	`[.0hik.1ko.0su]
tanin	`[0ta.0niN]
erai	`[.0e.1ra.0i]
zannen	`[.0zaN.1neN]
housou	`[0hO.0sO]

Code Samples

The following code samples illustrate how to use some of the IBM Text-to-Speech API's. Link with ibmeci.lib on Windows and libibmeci.a on AIX.

Hello world!

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();

   eciAddText(eciInstance, "Hello world!");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDelete(eciInstance);
   return 0;
}
```

Input from file

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();

   eciSynthesizeFile(eciInstance, "C:\\helloworld.txt");
   eciSynchronize(eciInstance);

   eciDelete(eciInstance);
   return 0;
}
```

Specifying a language

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNewEx(eciStandardFrench);

   eciAddText(eciInstance, "Bonjour le monde!");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDelete(eciInstance);
   return 0;
}
```

Specifying a voice

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();
   eciCopyVoice(eciInstance, 3, 0); // use Voice 3 (child)

   eciAddText(eciInstance,
      "I'm the only child in the IBM family.");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDelete(eciInstance);
   return 0;
}
```

Specifying a sample rate

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();
   eciCopyVoice(eciInstance, 3, 0); // use Voice 3 (child)
   eciSetParam(eciInstance, eciSampleRate, 0); // use 8k

   eciAddText(eciInstance, "Now my voice is lower quality.");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDelete(eciInstance);
   return 0;
}
```

Sample rates:

eciSampleRate = 0
8k (8000 Hz)
Lowest quality, but sufficient for telephony applications

eciSampleRate = 1
11k (11025 Hz)
Higher quality

eciSampleRate = 2
22k (22050 Hz)
Best quality, and compatible with desktop speech recognition

Specifying voice parameters

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();
   eciCopyVoice(eciInstance, 3, 0); // use Voice 3 (child)
   eciSetVoiceParam(eciInstance, 0, eciSpeed, 75);

   eciAddText(eciInstance, "I'm fast yet very accurate.");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDelete(eciInstance);
   return 0;
}
```

Using annotations

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();
   eciCopyVoice(eciInstance, 3, 0); // use Voice 3 (child)
   eciSetParam(eciInstance, eciInputType, 1); // annotations on

   eciAddText(eciInstance, "`vs75 Annotations are another");
   eciAddText(eciInstance, "way to change speed.");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDelete(eciInstance);
   return 0;
}
```

Concatenative TTS

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();
   eciCopyVoice(eciInstance, 1, 0); // use Voice 1 (adult male)
   eciSetParam(eciInstance, eciSampleRate, 0); // use 8k

   eciAddText(eciInstance, "Look, now I'm using");
   eciAddText(eciInstance, "a concatenative voice!");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDelete(eciInstance);
   return 0;
}
```

Inserting indices

```
#include <stdio.h>
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();
   eciRegisterCallback(eciInstance, f, 0);

   eciInsertIndex(eciInstance, 0);
   eciAddText(eciInstance, "hi");
   eciInsertIndex(eciInstance, 1);
   eciAddText(eciInstance, "there");

   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDelete(eciInstance);
   return 0;
}
```

Catching indices – the callback function

```
ECICallbackReturn f(ECIHand hEngine, ECIMessage Msg,
   long lParam, void *pData)
{
   switch (Msg)
   {
   case eciIndexReply:
      switch (lParam)
      {
      case 0: // about to say "hi"
         printf("hi\n");
         break;
      case 1: // about to say "there"
         printf("there\n");
         break;
      }
      break;
   }
   return eciDataProcessed;
}
```

User dictionaries – main volume

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();

   ECIDictHand dictHand = eciNewDict(eciInstance);
   eciSetDict(eciInstance, dictHand);
   eciUpdateDict(eciInstance, dictHand, eciMainDict,
      "world", "friends");

   eciAddText(eciInstance, "Hello world!");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDeleteDict(eciInstance, dictHand);
   eciDelete(eciInstance);
   return 0;
}
```

User dictionaries – roots volume

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();

   ECIDictHand dictHand = eciNewDict(eciInstance);
   eciSetDict(eciInstance, dictHand);
   eciUpdateDict(eciInstance, dictHand, eciRootDict,
      "program", "bowl");

   eciAddText(eciInstance, "I love programming!");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDeleteDict(eciInstance, dictHand);
   eciDelete(eciInstance);
   return 0;
}
```

User dictionaries – abbreviations volume

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNew();

   ECIDictHand dictHand = eciNewDict(eciInstance);
   eciSetDict(eciInstance, dictHand);
   eciUpdateDict(eciInstance, dictHand, eciAbbvDict,
      "ex.", "example");

   eciAddText(eciInstance, "One ex. is this.");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDeleteDict(eciInstance, dictHand);
   eciDelete(eciInstance);
   return 0;
}
```

User dictionaries – extended volume

```
#include <eci.h>

int main()
{
   ECIHand eciInstance = eciNewEx(eciStandardJapanese);

   ECIDictHand dictHand = eciNewDict(eciInstance);
   eciSetDict(eciInstance, dictHand);
   eciUpdateDictA(eciInstance, dictHand, eciMainDictExt,
      "__—t", "ƒYƒˆƒE", eciKoyuuMeishi);

   eciAddText(eciInstance, "__—t");
   eciSynthesize(eciInstance);
   eciSynchronize(eciInstance);

   eciDeleteDict(eciInstance, dictHand);
   eciDelete(eciInstance);
   return 0;
}
```

Glossary of Linguistic Terms

Alphanumeric symbols Alphabetic (a, b, c) and numeric (1, 2, 3) symbols.

Compound word A word created from two other words. Here are some examples of compound words:

hardwood, moon dancer, widespread, hand carry, tree trunk, overpower, outgrown, cow punch

A compound word can be combined with another word (which can be a compound), so there is no theoretical limit to the length of a compound:

firewood bin, graham cracker pie crust, greenhouse gas

English spelling does not indicate whether or not something is a compound. The component words can be separated with a space or a hyphen, or not separated at all:

free-fall, freeway, free will, highland, high-rise, high school

A compound word has a different stress pattern than a noun phrase consisting of the same words. For example, compare the pronunciation of the following:

Ouch! That's hard wood.

It's not a pine tree; it's a hardwood.

Please paint that black board yellow.

Please erase the blackboard this afternoon.

Content word The type of word that constitutes most of the vocabulary, such as:

- Nouns (*story*, *happiness*, *sun*, *mile*)
- Verbs (*ride*, *chew*, *listen*, *bring*, *believe*, *remain*)
- Adjectives (*brilliant*, *awful*, *three*, *new*, *darkest*)
- Adverbs (*often*, *far*, *much*, *calmly*, *happily*)

Content words are distinguished from function words.

Emphasis

Emphasis is the prominence given to a word relative to other words in an utterance.

Function word

Grammatical words such as:

- Conjunctions (*and, or, but*)
- Articles and determiners (*a, an, the, this, those...*)
- Auxiliaries (*can, may, will, must, should...*)
- Prepositions (*to, from, over...*)
- Pronouns (*she, her, we, they, it...*)

Function words are distinguished from content words and are normally pronounced with reduced emphasis.

Intonation

Changes in pitch across an utterance which are not related to the meaning of individual words. Intonation conveys, for example:

- The difference between questions and statements
- Contrastive emphasis, used in statements that contradict or parallel a previous statement (e.g., *Terry has a cold but JANET has pneumonia.*)
- Statement completion or closure

Intonational phrase

In IBM TTS, an intonational phrase is usually marked off by punctuation, such as a comma, period, or question mark.
One phrase: *He's a child?*
Two phrases: *He's a child, though growing quickly.*
Three phrases: *He's a child, an old child, but nevertheless a child.*

Key

A key is the first half of a user dictionary entry. The key is the string of characters that will be searched for by the dictionary routine.

Nuclear accent

The last emphasized word in an intonational phrase that has a degree of emphasis of 2 or higher.

Phonetic spelling A phonetic spelling uses special symbols like those found in the pronunciation guide of a dictionary. It has one symbol for each sound and indicates which syllables receive stress.

Pitch How high or low a voice sounds.

Reduced emphasis A word with reduced emphasis is shorter than normal and has no pitch accent (tone). A word that simply has *no emphasis* rather than *reduced emphasis* has no pitch accent, but it is not shortened.

Root The base form of a word, without prefixes (like *un-*) or suffixes (like plural *-s* or past tense *-ed*).

Stress Stress is the prominence given to a syllable, relative to other syllables in the word. For example, in sentence (a) the word desert has the greatest stress on the first syllable, and in sentence (b) the word desert has the greatest stress on the second syllable.
(a) “I've been through the desert in a car with no air conditioner.”
(b) “Let's desert this old car and walk from here.”

Syllable A syllable is a unit of speech containing, at a minimum, a sonorant nucleus such as a vowel or diphthong. The syllable may also contain one or more consonants surrounding the vowel.

Translation A translation is the second half of a user dictionary entry. The translation is the pronunciation or output specified by the user.

White space One or more spaces made with the spacebar or Tab key.

Appendix A Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only that IBM product, program, or service may be used.

Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service.

The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Asia-Pacific users can inquire, in writing, to the IBM Director of Intellectual Property and Licensing, IBM World Trade Asia Corporation, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106, Japan.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department T01B, 3039 Cornwallis, Research Triangle Park, NC 27709-2195, USA. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

IBM

ViaVoice

Adobe Acrobat is a trademark or registered trademark of Adobe Systems Incorporated.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation in the United States and/or other countries.

Microsoft, Windows, Windows NT, Windows 95, Windows98, and Windows 2000 logo are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Other company, product, and service names may be trademarks or service marks of others.

Index