festival-freebsoft-utils

for version 0.10

Milan Zamazal
Brailcom, o.p.s.

This manual is for festival-freebsoft-utils, version 0.10.

Table of Contents

1 Motivation

Festival is a powerful and extensible speech synthesis system, able to handle the whole text-to-speech process. The aim of festival-freebsoft-utils is to further extend Festival facilities, to the level providing complete set of features required by *Speech Dispatcher Manual*. As a side effect, festival-freebsoft-utils introduces interesting functionality, generalizing the text-to-speech system to a text-to-sound system.

Festival is well suited to the speech synthesis process itself, but lacks some end-user features, especially those needed for application sound output. festival-freebsoft-utils tries to fill this gap, thus making Festival suitable for screen readers and other speech output software, used especially by the blind and visually impaired people.

Main features of festival-freebsoft-utils are:

- Generalized concept of input events. festival-freebsoft-utils allows not only plain text synthesis, but also insertion of sounds and logical event mapping.
- Spell mode.
- Capital letter signalling.
- Punctuation modes, for reading or not reading punctuation characters.
- Function wrapping support.
- Speech Dispatcher Festival output interface. For more information about Speech Dispatcher, see `http://www.freebsoft.org/speechd` or *Speech Dispatcher manual*.

Up-to-date information about festival-freebsoft-utils can be found at its home page `http://www.freebsoft.org/festival-freebsoft-utils/`.

2 Installation

festival-freebsoft-utils was tested with Festival 1.4.3. Other versions of Festival may or may not work.

Having SoX (`http://sox.sourceforge.net`) installed is strongly recommended, many festival-freebsoft-utils functions don't work without it.

As Festival does not support UTF-8 encoding, festival-freebsoft-utils uses the iconv utility for character coding conversions. iconv is a standard part of some (e.g. GNU) operating systems, if you don't have it you can install it as a part of the libiconv library available at `http://www.gnu.org/software/libiconv/`.

Installation itself is easy, just copy all the `*.scm` files to one of the directories present in the Festival's load-path. This is typically `/usr/share/festival/`, you can get the exact list of the directories by evaluating `load-path` in the Festival command line interface. Then you can load the whole system at Festival startup by adding the line

```
(require 'speech-dispatcher)
```

to the Festival initialization file (typically `/etc/festival.scm` system wide or `~/.festivalrc` for a particular user). Note you needn't do this for Speech Dispatcher operation as Speech Dispatcher invokes this call itself.

3 User Customization

You can customize festival-freebsoft-utils through several user variables described in the following sections. This chapter is primarily focused on Speech Dispatcher users and other users not using festival-freebsoft-utils directly. For a complete usage description see Chapter 4 [Reference Manual], page 7.

Most of the extensions presented here don't work with usual Festival functions such as `SayText`, which are too limited in their use. You must use either the Speech Dispatcher functions, Section 4.15 [speech-dispatcher.scm], page 17, or the event speaking functions, Section 4.4 [events.scm], page 11, to utilize the features like word substitution, capital signalling or punctuation modes.

3.1 The Concept of Events

Default Festival interfaces expect text on their inputs, either in a plain form or in the form of some markup. The event module generalizes the concept of input to *events*. Event is a general input object that can represent not only text, but also a pre-defined sound or an object just mapping to another input object.

Currently the following kinds of events are supported:

`text` Text. The event value is a string containing the given text.

`ssml` Text represented in the SSML 1.0 markup. The event value is a string containing the given text. Please note that festival-freebsoft-utils does not provide conforming SSML implementation and implements only a limited subset of the standard. Moreover, due to the limitations of the built-in Festival XML parser, SSML markup texts must be provided in the target encoding of the document languages.

`sound` Sound icon. The value is a string containing a file name of a sound file, either absolute, or relative to the `sound-icon-directory` variable value.

`character`
Single character. The value is a string containing the character.

The difference between character events and text events is that characters may be spoken in a different way than ordinary single-letter texts.

`key` A key (as on keyboard). The value is a string containing key description in the format defined by the Speech Synthesis Interface Protocol, *SSIP*.

`logical` Symbolic event name, usually mapped to another event. The value is an arbitrary symbol. Logical event values starting with the underscore character are considered special and shouldn't be generally used. See Section 4.4 [events.scm], page 11, for more details.

Any event may be mapped to another event. Before festival-freebsoft-utils functions process an event, they check for its mapping and if the event is mapped to another event, it is replaced by the target event. Event mapping is defined by the following variable:

`event-mappings`
Maps events of any supported kind (i.e. logical, text, sound, character, key) to other events (typically to text or sound events). All logical events used must be

defined here, other kinds of events are processed in some default way if there are not defined in this variable.

The variable contains an alist whose elements are of the form (*event-type mappings*). *event-type* is one of the symbols `logical`, `text`, `sound`, `character`, `key`. *mappings* is an alist with the elements of the form (*value new-event-type new-event-value*).

See the default variable value for an example.

For a convenience, there is a function that allows you to add or replace single event mappings in the `event-mappings` variable in an easier way than redefining the whole variable value:

`set-event-mapping!` *event-type event-value new-event-type* `new-event-value`
Ensure the event of *event-type* and *event-value* is mapped to the event of *new-event-type* `new-event-value`.

Example:

```
(set-event-mapping! 'logical 'hello 'text "Hello, world!")
```

3.2 Word Substitution

You can map words to events. This is useful especially when you want to replace some words by sounds.

`word-mapping`
Alist mapping words to events. Each entry of the list is of the form ("*word*" *event-type event-value*). If *word* is encountered in the input text, it is replaced by the given event.

3.3 Signalling Capital Characters

When capital character signalling is enabled, Section 4.6 [cap-signalization.scm], page 11, capital characters are signalled via the `capital` logical event. By default the event is mapped on the sound event `capital`. If you want to change it, change the logical mapping, as is described in Section 3.1 [Events], page 3.

For example, the following code in your `~/.festivalrc` changes the sound signalling to saying the word *capital*:

```
(require 'events)
(set-event-mapping! 'logical 'capital 'text "capital")
```

3.4 Reading Punctuation Characters

Through the punctuation modes, Section 4.7 [punctuation.scm], page 12, you can force Festival to speak all punctuation characters. Since the default English voices don't have defined pronunciation of some punctuation characters, it is provided through the following variable:

`punctuation-pronunciation`
Alist of punctuation characters and their word forms. Each entry in the list is of the form ("*character*" "*word*" ..., where *character* is the pronounced

character and *words* are the words of its pronunciation. Please note you must put each word inside separate double quotes. Example entry:

```
("!" "exclamation" "mark")
```

3.5 Avoiding Initial Pauses

Festival inserts initial pause in each synthesized utterance. There is a good reason for it—speech starts from silence and thus the first diphone of the synthesized sample should be *pause-first phoneme*.

However there are some situations when the initial pause is undesirable. For instance, when reading characters in a speech enabled editor, the initial pauses slow down the reading. So festival-freebsoft-utils provide a way to disable the initial pause by making its effective duration zero.

First, you must load the corresponding code:

```
(require 'nopauses)
```

After this, there is a variable available controlling the initial pause insertion:

`inhibit-initial-pauses`
: When set to a non-`nil` value, initial pauses are inhibited.

3.6 Tokenization

If you use the festival-freebsoft-utils tokenizer instead of the Festival built-in tokenizer, you can put additional limits on the tokenization process besides the `eou_tree`.

`max-number-of-tokens`
: Maximum number of tokens in a single utterance. Utterance chunking is performed in such a way that each produced utterance contains at most this number of tokens.

`max-number-of-token-chars`
: Maximum number of characters within a single token. If a token contains more characters than is stated by this limit, it is split into smaller tokens.

3.7 Voice Selection

You can configure languages and voices used by the SSML, Speech Dispatcher and other interfaces supporting the mechanism with the following variables:

`language-codes`
: Alist mapping ISO language codes to Festival language names. Each alist entry is of the form (`"`*`language-code`*`"` *`language-name`*), where *language-code* is an ISO language code as used by Speech Dispatcher and `language-name` is the corresponding Festival language name. Optionally, the alist elements can have the extended form (*`language-code language-name . dialects`*), where *dialects* is a list of pairs (*`dialect-code dialect-name`*). *dialect-code* is the part of the language code after a hyphen or underscore and *dialect-name* is the dialect name used by Festival voices.

`voice-select-defaults`
: Alist of default voice parameters. Each alist entry is of the form `(name value)`, where *value* can be either the actual parameter value or `nil`, meaning the value is unspecified.

3.8 Using with Speech Dispatcher

One of the primary goals of festival-freebsoft-utils is to serve as a Speech Dispatcher interface to Festival. festival-freebsoft-utils is required by Speech Dispatcher for the use of Festival as the speech synthesis backend.

In order to use festival-freebsoft-utils with Speech Dispatcher, you need not to make any special festival-freebsoft-utils arrangements. Just configure it as is described in the previous sections. It is particularly recommended to configure available languages if you want to use Festival for other languages than English, See Section 3.7 [Voice Selection], page 5.

This version of festival-freebsoft-utils requires Speech Dispatcher 0.5 or higher.

4 Reference Manual

festival-freebsoft-utils consists of several modules described in the following sections.

4.1 util.scm

This module contains miscellaneous utilities useful in general SIOD and Festival programming.

Macros and functions mostly available in Lisp dialects:

`when` *condition body-form ...*
: If and only if *condition* is true, evaluate `body-form`s.

`unless` *condition body-form ...*
: If and only if *condition* is false, evaluate `body-form`s.

`prog1` *form ...*
: Evaluate all forms and return the return value of the first one.

`let*` *bindings body-form ...*
: The same as `let` except that variable bindings are applied sequentially rather than in parallel.

`unwind-protect*` *protected-form cleanup-form ...*
: Evaluate *protected-form*, and after it is finished, whether successfully or with errors, evaluate all *cleanup-form*s. If *protected-form* was evaluated successfully, return its return value.

  Unlike Festival's `unwind-protect`, `unwind-protect*` accepts multiple *cleanup-form*s and evaluates them even when *protected-form* doesn't signal an error.

`first` *list*
`second` *list*
`third` *list*
`fourth` *list*
: Return first, second, third or fourth element of *list*, respectively.

`butlast` *list*
: Return the *list* without its last element. If *list* is empty, return an empty list.

`min` *x y* Return minimum of the two numeric values *x* and *y*.

`max` *x y* Return maximum of the two numeric values *x* and *y*.

`abs` *x* Return absolute value of *x*.

`remove-if` *test list*
: Return *list* with elements, for which the *test* call returns non-`nil`, removed. The order of list elements is preserved. *test* must be a function of a single argument.

`identity` *object*
: Return *object*.

`complement` *`function`*
: Return a function that is equivalent to function except that it returns the opposite truth value to the return value of function.

`apply*` *`function arglist`*
: The same as `apply`, except that it also works if *function* is given as a string.

`dolist (`*`var items`*`)` *`body-form ...`*
: Loop over *items* and perform *body-form*s over each of them, binding it to the variable *var* (unevaluated).

`add-hook` *`hook-variable hook-function to-end?`*
: Add *hook-function* to *hook-variable* if it is not already present there. *hook-variable* must be a variable containing a list. If *to-end?* is true, add *hook-function* to the end of the list contained in *hook-variable*, otherwise add it to the beginning.

`assoc-set` *`list key value`*
: Add the *key-value* pair to the association `list` and return the resulting list. Contingent previously *list* entries stored under *key* are removed from the resulting list.

`avalue-get` *`key alist`*
: Find the first *alist* element of the form `(`*`key*`* `value)`, where *key** is `string-equal` to *key*, and return *value*.

`avalue-set!` *`key alist value`*
: Destructively set the *value** of the first *alist* element of the form `(`*`key*`* `value*)` to *value*. Return *alist*.

`avg .` *`args`*
: Return average value of *args*.

`dirname` *`path`*
: Return the directory part of *path*.

`make-temp-filename` *`template`*
: Return name of a (probably non-existent) temporary file. *template* is a basename of the file, that is formatted with the `format` function and must contain exactly one `%s` sequence to be replaced with a variable part of the file name.

  Actually, this function is somewhat limited by the available Festival system interface. So it is not safe, the file may be created before it is actually used or the function may fail with an error. But for simple purposes the function should work fine and it shouldn't be worse than the standard `make_tmp_filename` function.

`with-temp-file` *`filename body`*
: Macro that binds newly generated temporary file name to a local variable *filename* and then performs *body*. The macro ensures the temporary file is deleted after finishing *body* in any way.

`string-replace` *`string from to`*
: Replace all occurrences of *from* by *to* in *string* and return the result.

Festival specific utilities:

`item.has_feat` *item feature*
: Return true if and only if *item* has *feature* set.

`langvar` *symbol*
: Return language dependent value stored under *symbol*. First, the variable named *symbol.language*, where *language* is the language name as stored in the `Language` parameter is checked and if it is unbound, *symbol*'s value is returned.

`current-voice-coding`
: Return character coding of the currently selected voice in Festival. The coding is taken from the `coding` attribute of the voice description, if it is undefined or `nil`, ISO 8859-1 coding is assumed. The `coding` voice attribute is introduced by festival-freebsoft-utils, it is not a standard Festival feature.

`utt-relation-top-items` *utt relation*
: Return a list of top level items in *relation* in `utt`.

`do-relation-items (`*var utterance relation*`)` *body-form ...*
: Loop over *relation* items of *utterance*, performing *body-form*s for each of them, binding it to the variable *var*. The macro arguments *var* and *relation* are not evaluated.

`do-relation-top-items (`*var utterance relation*`)` *body-form ...*
: Similar to `do-relation-items`, but loops only over the relation's top items.

4.2 Wave form handling

There are some utility functions to help handling wave forms:

`wave-concat` *waves*
: Append wave forms and return the resulting wave form. *waves* must be a list of wave forms to append.

`wave-subwave` *wave from-time to-time*
: Return the part of *wave* form that starts at *from-time* and finishes at *to-time*. Both times are in seconds.

`wave-load` *filename*
: Load and return a wave form from *filename*. This function is similar to `wave.load`, but more sound file formats (most significantly Ogg Vorbis, if your SoX installation supports it) can be loaded.

`wave-utt` *wave*
: Create and return an utterance, that contains just the `Wave` relation holding *wave*.

`wave-import-utt` *filename*
: Create and return an utterance, that contains just the `Wave` relation holding a wave loaded from *filename* via the function `wave-load`.

4.3 oo.scm

Sometimes it is useful to extend a Festival function in some way. Standard Festival functions don't provide easy to use means for it. This module tries to fill the gap.

The following macro allows you to wrap a defined function:

`define-wrapper (`*function arg ...*`)` *wrapper-name . body*)
: Wrap *function* with arguments *arg ...* by the code *body*. Given function arguments must match the arguments of the wrapped function. *wrapper-name* is a symbol uniquely identifying the wrapper, it allows redefinition of the wrapper. One function can be wrapped by any number of wrappers. None of the `define-wrapper` arguments is evaluated.

  Within *body*, a function named `next-func` is automatically defined. It returns the next wrapper or the original function. Please note `next-func` is a function, not a variable, so its typical invocation looks as follows: `((next-func)` *arg ...*`)`.

`oo-ensure-function-wrapped` *function-name*
: If a wrapped function gets redefined, its wrapper is lost. If you want to ensure the function is still wrapped before its use, you may call this function, with its symbol name as the argument.

`oo-unwrapped` *function-name*
: Return the original definition of a wrapped function.

Example use:

```
festival> (define (foo x) (+ x 42))
#<CLOSURE (x) (+ x 42)>
festival> (foo 1)
43
festival> (define-wrapper (foo x) my-foo-wrapper (print "Foo called.") ((next-func) x)
nil
festival> (foo 1)
"Foo called."
43
```

You can also wrap parameters, set by `Param.set`:

`Param.wrap` *name wrapper-name . body*
: Wrap access to parameter *name* by code *body*. If the given parameter is accessed, its wrapper is invoked instead of just returning the parameter value. *wrapper-name* is the same as in `define-wrapper`.

  Macro `next-value` is automatically defined within *body*. It returns the parameter value, either plain or modified by another wrapper.

Example use of parameter wrapping:

```
festival> (Param.set 'foo 42)
#<feats 0x8169950>
festival> (Param.wrap foo foo-w (+ (next-value) 1))
nil
```

```
festival> (Param.get 'foo)
43
```

And finally, the `glet*` macro allows you to dynamically bind a global variable value:

`glet*` *bindings . body*
: Similar to `let*` (see Section 4.1 [util.scm], page 7) except that the variables in *bindings* are bound dynamically instead of lexically. All variables in *bindings* must be global variables.

4.4 events.scm

For introductory and configuration information about events see Section 3.1 [Events], page 3. The event module provides the following functions to synthesize events:

`event-synth` *type value*
: Synthesize event of *type*, which may be one of the following symbols: `logical`, `text`, `sound`, `character`, `key`. *event* is the event value that must correspond to the event type.

`event-play` *type value*
: Play event. The *type* and *value* arguments are the same as in `event-synth`.

Logical events starting with underscore are reserved for special purposes. Currently, the following special purpose logical events are recognized:

`_debug_on*`
: Turn on debugging. That means every processed event is logged. `_debug_off` is just a prefix, it can be followed by any symbol constituent characters.

`_debug_off*`
: Turn the debugging off. `_debug_off` is just a prefix, it can be followed by any symbol constituent characters.

4.5 spell-mode.scm

Defines spelling mode, i.e. the mode in which the input text is spelled rather than read in the usual way. The `spell` mode is a normal Festival mode, so you can use it after loading this module immediately, e.g.

```
(tts_file "file" 'spell)
```

4.6 cap-signalization.scm

Defines mode that allows signalling of capital letters through the logical event `capital`. See Section 3.3 [Capital Letters], page 4, for more details.

`set-cap-signalization-mode` *mode*
: If *mode* is true, enable capital letter signalling, otherwise disable it.

4.7 punctuation.scm

Sometimes it is useful to get read all the punctuation characters present in the synthesized text (for exact information about the text) and sometimes it is useful to read no punctuation character (for faster reading). Punctuation modes allow you to tell Festival, whether it should read punctuation characters or not.

`set-punctuation-mode` *mode*
: Set punctuation mode to *mode*. *mode* may be one of the following symbols: `all` meaning all the punctuation characters are read, `none` meaning no punctuation characters are read, and `default` that switches to the default Festival behavior corresponding to the current language and voice.

See Section 3.4 [Punctuation Characters], page 4, for information about punctuation mode configuration.

4.8 tokenize.scm

Festival's tokenization is implemented mostly in C++, so it is impossible to use it when extending Festival. The `tokenize` module provides an alternative tokenization implemented in SIOD, that can be used wherever needed.

`next-chunk` *text*
: Get the next part of *text* and create an utterance containing the corresponding tokens. A list of two elements, the utterance and the remaining unprocessed part of *text*, is returned.

An alternative `SayText` function, splitting the text into smaller pieces (and thus speeding up the start of speech) might be implemented as follows:

```
(define (SayText* text)
  (if (not (equal? text ""))
      (let ((utt-text (next-chunk text)))
        (let ((utt (car utt-text))
              (text (cadr utt-text)))
          (utt.play (utt.synth utt))
          (SayText* text)))))
```

4.9 multiwave.scm

Sometimes it is convenient to return multiple synthesized wave forms instead of a single wave form. There are two typical situations when this can happen:

- You want to synthesize a long text and you don't want to wait until it is all synthesized, you want to play the resulting audio as soon as possible. The text can be cut into smaller pieces, returning the corresponding wave forms in a sequence.
- The resulting wave form is a mixture of a synthesized texts and sound icons, of different rates or other sound sample parameters. Concatenating them together may reduce the resulting sound quality. So the different sound parts may be better returned separately.

The `multiwave.scm` module provides the following interface for those purposes:

`multi-synth` *`type value`*
: This function is similar to the `event-synth` function (see Section 4.4 [events.scm], page 11), except that it doesn't return an utterance containing the resulting wave form. Instead, it setups the synthesis for the following `multi-next` calls.

`multi-next`
: Return the next wave form of the last event synthesized via the `multi-synth` function. If there is no next wave form, return `nil`.

  If you synthesize an SSML text, the function may return a non-`nil` symbol instead of a wave form. Then the symbol is a name of the mark just reached.

`multi-clear`
: Throw away the synthesized data. You usually don't need to call this function as the data is cleared on the next `multi-synth` call automatically, but the function may be useful under special circumstances.

4.10 voice-select.scm

The voice-select module provides a mechanism for voice and language selection. For its configuration information, see See Section 3.7 [Voice Selection], page 5.

The following voice selection functions are available:

`voice-list`
: Return the list of names of all available voices. Unlike the list returned by the standard Festival function `voice.list`, the list includes all registered voices.

`voice-list-language-codes`
: Return the list of names, language codes and dialect codes of all available voices. Each element of the returned list is of the form (*`name language-code dialect-code`*), where all the elements of the tripple are symbols. If the language code or the dialect code is not known for the voice, the corresponding element is `nil`.

`current-language-voices`
: Return a list of all the voices available for the current language.

`select-voice` *`language dialect gender age variant name`*
: Select voice according to the specified parameters and return its name. *language* is the language name, *dialect* is a *language* dialect name. *gender* can be one of the symbols `male` or `female` (the value is currently ignored). `age` is age of the speaker in years given as a number. `variant` is a positive integer that selects one of several voices, if more than one voice is selected by all other parameters. *name* can be a particular voice name.

  Each of the function arguments can have `nil` as its value. In such a case, default value of the corresponding parameter is used. If there is no default value, the parameter is not considered in the selection process.

  If more than one voice matches, one of the matched voices is selected. If no voice satisfying all the given parameters is available, some voice satisfying the most important parameters is selected.

`select-voice* lang-code gender age variant name`
: Like `select-voice`, except that language and dialect are specified by an ISO *lang-code*.

`reset-voice`
: Reset currently selected voice parameters to their default values.

Additionally, there is a variable holding information about voice properties currently in effect for the purpose of voice selection by the `voice-select` function:

`voice-select-current-defaults`
: Alist containing current voice properties used by voice selection.

4.11 prosody-param.scm

Ever wished to be able to change prosodic parameters in Festival easily and in a uniform way for different voices? Well, here are the appropriate functions. They are no way guaranteed to work for all voices, since each voice can have its own unique way of prosody handling. But they should work for typical cases.

`set-pitch value`
: Set mean pitch of the voice to *value* and return the old pitch value. *value* is given in Hertz.

`set-pitch-range value`
: Set the pitch range of the voice to *value* and return the old pitch range value. The value is in percents of the mean pitch, its clear meaning is not defined.

`set-volume value`
: Set volume to the given *value* and return the old volume value. *value* must be in the range 0–1 from silence to maximum (the default).

`set-rate value`
: Set voice rate to *value* and return the old rate value. *value* is in the range 0.1–10 from the slowest to the fastest. The value 1 corresponds to the normal voice speed, other values multiply the voice speed appropriately.

The *value* argument of all the functions above may also be a function of a single argument accepting the current parameter value and returning its new value. The following convenience functions return functions which adjust the parameter values appropriately:

`prosody-shifted-value shift`
: Return a function modifying the value by adding *shift* to it.

`prosody-relative-value coef`
: Return a function modifying the value by multiplying it by *coef*.

Example setting doubling the current voice speed:

```
(set-rate (prosody-relative-value 2))
```

When you switch to a different voice, prosody parameters get lost. festival-freebsoft-utils offers a way to restore them, using the following functions:

`change-prosody` *function value*
: Similar to the `set-*` functions described above, except it additionally saves the set prosodic value. *function* is one of the `set-*` functions and *value* is its parameter value.

`restore-prosody`
: Set prosodic parameters according to their current saved values.

`reset-prosody`
: Reset the list of the saved prosodic values. Note, it just deletes the saved settings and doesn't actually change the current prosodic parameters.

4.12 ssml-mode.scm

festival-freebsoft-utils provides a text processing mode partially supporting the SSML 1.0 markup. You can process a SSML file in the following way in the Festival prompt:

```
(tts_file "/the/path/to/the/file" 'ssml)
```

Moreover, there are some particular SSML processing functions available:

`ssml-say` *text*
: Speak the given SSML *text*.

`ssml-parse` *text*
: Parse the given SSML *text* for later processing by the `ssml-next-chunk` function.

`ssml-next-chunk`
: Return next utterance containing tokens from the last SSML text parsed by `ssml-parse` or a mark name. If there is no next utterance or mark, return `nil`. The returned utterance contains only the `Token` relation and is intended to be further processed with the `utt.synth` function. The returned mark name is a non-`nil` symbol.

`ssml-speak-chunks`
: Speak the SSML text processed by `ssml-parse`.

The `ssml-parse` and `ssml-next-chunk` functions are intended to be used when you want to synthesize an SSML text, but not to speak it immediately. A sample use of those functions for a hypothetic `ssml-say*` function similar to `ssml-say` might be as follows:

```
(define (ssml-say* text)
  (ssml-parse text)
  (ssml-say*-1))

(define (ssml-say*-1)
  (let ((utt (ssml-next-chunk)))
    (if utt
        (begin
          (cond
           ((symbol? utt)
            (print utt))
```

```
 (utt
  (utt.play (utt.synth utt))))
(ssml-say*-1)))))
```

If you need quick synthesizer response, avoid the DOCTYPE declaration in your SSML data. The DOCTYPE declaration takes an observable time when processed in the Festival's XML parser.

festival-freebsoft-utils does not provide conforming SSML implementation and implements only a limited subset of the standard. Moreover, due to the limitations of the built-in Festival XML parser, SSML markup texts that contain non-ASCII characters can only be processed with the `ssml-parse` and `ssml-next-chunk` functions and must be provided in the UTF-8 encoding.

It is not easy to fully support SSML in Festival. Contingent support and contributions are welcome.

4.13 fileio.scm

Functions in this module try to help to improve Festival file input/output.

`with-open-file (`*variable filename* `&optional` *how*`) `*body-form* `...`
: Open file named *filename* in mode *how* (for reading if not provided) and bind it to a newly created local *variable*. Execute *body-form*s in that context.

`read-file` *filename*
: Read and return whole contents of the file named *filename*.

`write-file` *filename string*
: Write *string* to a file named *filename*.

`make-read-line-state`
: Create and return a state object required by the function `read-line`.

`read-line` *file state*
: Read a single line from *file* and return it without the final newline character. If there is no next line in *file*, return `nil`. *state* is a state as initially returned by the `make-read-line-state` function.

  The typical `read-line` usage idiom is:

```
(let ((state (make-read-line-state))
      (f (fopen "..."))
      (line t))
  (while line
    (set! line (read-line f state))
    ...))
```

4.14 recode.scm

Festival doesn't support different character sets directly. But it is 8-bit clean and you can use whatever character coding you like if you can process it in the form of 8-bit characters. The `recode.scm` module offers the following functions to convert between different character sets:

`recode` *`string from-coding to-coding`*
: Return given *string*, originally encoded in *from-coding*, recoded to *to-coding*.

`recode-utf8->current` *`string`*
: Return given *string*, originally encoded in `UTF-8`, recoded to the coding of the current voice.

  Before applying normal recoding this function translates strings as specified in `recode-special-utf8-translations` variable. This allows you to convert some unicode characters in a special way, e.g. to translate empty space to space (thus separating words around it). `recode-special-utf8-translations` contains lists of two elements, the converted substring and its translation.

The recoding functions use the `iconv` program and temporary files to convert between character sets. There is no known better way to do the conversions.

4.15 speech-dispatcher.scm

This module provides Speech Dispatcher interface. You need it if you want to use Festival as a Speech Dispatcher output text-to-speech system. The module defines functions required by the Speech Dispatcher Festival output module and user configuration variables, see Section 3.8 [Speech Dispatcher], page 6.

To ease debugging, for each Speech Dispatcher function which returns a wave form, there is defined a corresponding function of the same name with star appended, that returns an utterance instead of wave form. For instance, the function `speechd-speak` returns a wave form (and can be used only in server mode), while the function `speechd-speak*` returns an utterance.

5 How to contact us

The author of festival-freebsoft-utils is Milan Zamazal `pdm@freebsoft.org`. The home page of festival-freebsoft-utils is `http://www.freebsoft.org/festival-freebsoft-utils/`.

You can contact us with your comments, questions, suggestions, patches or anything at the Speech Dispatcher mailing list `speechd@freebsoft.org`. Bug reports can be sent to the e-mail address `festival-freebsoft-utils@bugs.freebsoft.org`.

festival-freebsoft-utils is part of the Free(b)soft project aimed at making computers accessible to blind and sorely visually impaired people. The home page of the project is `http://www.freebsoft.org`.

Appendix A GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

N

O

P

R

S

T

U

V

W