Spelling Chequer API

version 0.1 January 27, 1997

1.0 Introduction

The spell-checker is a fast, dictionary-based spell-checker. It supports letter insertion, letter deletion, letter transposition, letter substitution, and phonetic substitution. It can also split run together words. It is not as robust as some other spell-checkers in terms of allowing combinations of the above operations: it only allows a single insertion/deletion/transposition/substitution per word. But in terms of phonetic substitution, it allows a slightly wider range of phonetic substitutions to be made, which is probably more appropriate for young spellers.

It remembers words that have been skipped, so it won't ask you about them again, and it supports an interface for learning words.

It is also smart about locales. We have moved locale-specific spellings (color, colour) to locale-specific dictionaries, so these spellings will only be allowed in the proper locale.

2.0 API Description

The spelling checker API is an optional add on to Newton 2.1 OS. The API consists of a number of global functions. There are no user interface prototypes (all the ones in the Word Processor are part of the that package, not part of spell checker).

As of this writing the spelling checker API will be on the English Language versions of the MessagePad 2000 and eMate 300. Before using the spelling checker you should make sure it is installed. You can do this by checking for the existence of any of the spelling checker global functions:

```
if GlobalFnExists('SpellDocBegin) then
      // spelling checker installed
else
      // spelling checker not installed
```

The spell-checker has a very simple interface. Tell the spell-checker that you're about to start spell-checking a document by calling `speller := SpellDocBegin()`. Then for each word in the document, you call `SpellCheck(speller, word)`. SpellCheck will return non-nil if the word appears to need correction. Then you call `SpellCorrect(speller, word)`. SpellCorrect will return a list of possible alternate guesses. If you want the spell-checker to remember words that have been skipped, call `SpellSkip(speller, word)`. If you want a word to be learned, call `SpellLearn(speller, word)`. When you have finished processing all the words in the document, call `SpellDocEnd(speller)`. This clears out the list of words that have been skipped, and it causes the learned words to be saved to the user store.

2.1 Processing of words passed to the spell-checker

Many of the spell-checking functions accept a word as a parameter. This parameter is processed similarly by these function.

1. Strip leading and trailing punctuation from the word
2. Convert the word to lower-case remembering original capitalization
3. If the word contains illegal symbols halt processing

The legal symbols are all alphanumeric characters (A-Z, a-z, 1-0) and the following special and punctuation characters: ! " ' ( ) : ; ? , . “ ” ‘ ’

2.2 Use of dictionaries by the spell-checker

The spell-checker assembles a list of built-in dictionaries to use for both checking and correction. The exact list of dictionaries used will vary from product to product, so the result of spell-checking will also vary. Generally, though, a word is considered to be valid if it is one of the built-in dictionaries or the Personal Word List or the list of skipped words.

2.3 The Spell Check Object

Most calls to the spell check API require passing an initialized spell check object. The object is initialized using the SpellDocBegin call. The object is returned as a frame. Although the object contains several slot, the only documented slots are:

`countSkippedAsMisspelled`

Determines if words on the skip list should be counted as misspelled. See the documentation for `SpellSkip` below (section 2.4).

`dialogInhibit`

Enable or disable the display of the capitalization slip. This slip can be displayed by `SpellLearn`, if the learned word is capitalized. See `SpellLearn` below (section 2.4).

2.4 The APIs

`speller := SpellDocBegin();`

speller spell check object used for other spell check calls (a frame)

Initializes the spell-check object and prepares it for use. You must call this function before spell-checking a document. It should always be balanced by a call to `SpellDocEnd`.

`SpellDocEnd(speller)`

speller spell-check object returned by `SpellDocBegin`

Call this function when you have finished a spell-checking session. If you used the `SpellLearn` function this function will save the learned words to the personal word list. The function will also delete the list of skipped words and deallocate the spell check object.

Important: Once you have called `SpellDocEnd` on a spell check object, do not attempt to use that object again. Doing so could result in bus errors or other errors.

`SpellCheck(`*`speller, word`*`)`

speller a spell checking object previously initialized using `SpellDocBegin`

word `the word to check passed as a NewtonScript string`

This function first processes the word as described above. If it contains invalid characters, then `SpellCheck` returns nil. It then looks to see if the word is in the dictionary list. If so, `SpellCheck` returns nil. If the word is not in a list, or if its capitalization is not correct, then `SpellCheck` returns non-nil.

For example:

```
SpellCheck("and") => nil      // spelled correctly
SpellCheck("And") => nil      // spelled correctly
SpellCheck("AND") => nil      // spelled correctly
SpellCheck("ernie") => 128    // capitalization is wrong
SpellCheck("Ernie") => nil    // spelled correctly
SpellCheck("ERNIE") => nil    // spelled correctly
SpellCheck("ibm") => 192      // needs all caps
SpellCheck("Ibm") => 192      // needs all caps
SpellCheck("IBM") => nil      // spelled correctly
SpellCheck("(and.)") => nil   // spelled correctly
SpellCheck("(bxnd.)") => true // not spelled correctly
SpellCheck("isn't") => nil    // spelled correctly
SpellCheck("isn’t") => nil    // spelled correctly
SpellCheck("so-so") => nil    // hyphen an invalid character
SpellCheck("ab#de") => nil    // invalid characters
SpellCheck("so") => nil       // spelled correctly
```

Important: Do not rely on a particular non-nil return value from `SpellCheck`. You can only rely on the value being either NIL or a non-nil.

`SpellCorrect(`*`speller, word`*`)`

speller a spell checking object previously initialized using `SpellDocBegin`

word `the word to check passed as a NewtonScript string`

This function first processes the word as described above (see section 2.1). It then uses the resulting word to generate a list of possible guesses for the word. If it finds no guesses, it either returns nil or an empty array of guesses. The returned list will contain up to 7 guesses, ordered generally by their closeness to the original word. Each returned guess will be punctuated as was the original word, and it will also be capitalized like the original (unless fixing the capitalization was part of the problem), so the returned guesses can be directly substituted for the original word.

Note: `SpellCorrect` can suggest alternates for correctly spelled words, even though it is normally only used for words that `SpellCheck` flagged as incorrect.

The following examples provide an indication of the types of corrections that the spell-checker will make.

```
// letter substitution
SpellCorrect("bxnd") => ["band", "bend"...]

// letter deletion
SpellCorrect("baxnd") => ["band"]

// letter insertion
SpellCorrect("bnd") => ["bond", "band"...]

// capitalization
SpellCorrect ("ernie") => ["Ernie", "Renie"...]

// fancy apostrophe preserved
SpellCorrect ("hisn't") => ["hasn't", "isn't"]

// punctuation preserved
SpellCorrect("(and.)") => ["(and.)", "(end.)"...]

// words split
SpellCorrect("looseends") => ["loose ends"]

// phonetic substitution
SpellCorrect("unfourtaneatly") => ["unfortunately"]

// case preserved
SpellCorrect("Shes") => ["She's", "Shoes"...]

// case preserved
SpellCorrect("SHEP") => ["SHEEP", "SHIP"...]
```

SpellSkip(*speller*, *word*)

speller a spell checking object previously initialized using `SpellDocBegin`

word `the word to check passed as a NewtonScript string`

`SpellSkip` is used to maintain a list of words that should be skipped during the course of spell-checking a document, but that should not be added permanently to the Personal Word List. `SpellSkip` processes the word as described above (see section 2.1). It then adds the word to the list of skipped words. `SpellSkip` ignores capitalization: it just stores the word as written.

Normally `SpellCheck` will return NIL if the word is found in the skip dictionary. However this doesn't let you keep track of how many words were skipped. To do this, there's a special slot that you can put into the spell check frame returned by `SpellDocBegin`. If you want to count skipped words as misspelled, set `speller.countSkippedAsMisspelled` to a non-nil value. When `SpellCheck` returns a non-nil value, set the slot to nil, and call `SpellCheck` again. If the result is nil, the word is in the skip dictionary and should be skipped.

SpellLearn(*speller*, *word*)

speller a spell checking object previously initialized using `SpellDocBegin`

word `the word to check passed as a NewtonScript string`

`SpellLearn` is used to add words permanently to the Personal Word List. Before calling SpellLearn, you should have first called `SpellDocBegin`. First `SpellLearn` processes the word as described above (see section 2.1). It then adds the word to the Personal Word List as described below.

For a capitalized word, or a word that is all caps, `SpellLearn` will display a notification slip that asks the user whether the word should always be capitalized (or all caps). If the user answers "Yes", it will be added as originally written, otherwise it will be converted to lower-case before being added.

The notification slip can be disabled by setting `speller.dialogInhibit` to non-nil.

`SpellLearn` returns the word that was actually learned, so that you can pass it to `SpellUnlearn` for implementing undo operations.

`SpellUnlearn(`*`speller, learnedword`*`)`

speller a spell checking object previously initialized using `SpellDocBegin`

word `the word to remove from the learned words list passed as a NewtonScript string`

`SpellUnlearn` is used to remove words from the Personal Word List, and is usually used to implement undo operations. The following example shows how SpellUnlearn is typically used:

```
learnedWord := SpellLearn(sp, word);
SpellUnlearn(sp, learnedWord);
```

Note: `SpellUnlearn` should be passed the word returned by the `SpellLearn` call, not the original word that was passed to `SpellCheck` or `SpellLearn`.