AlphaSpell

Fergus Duniho

COLLABORATORS			
	TITLE : AlphaSpell		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Fergus Duniho	August 23, 2022	

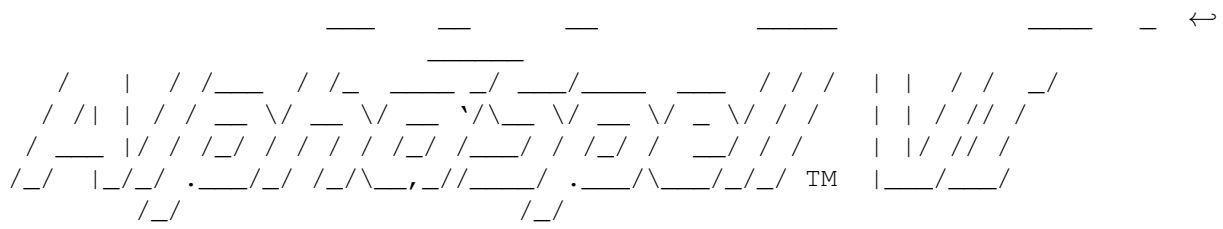
REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AlphaSpell

1.1 AlphaSpell VI



Copyright © 1992-6 Fergus Duniho
fdnh@troi.cc.rochester.edu

1.2 Introduction

AlphaSpell VI is a spelling checker that is designed to work alongside your text editor with the help of ARexx. Through the help of an ARexx script, you can spell check the currently activated document in your text editor. AlphaSpell comes with ARexx scripts for interfacing with several different text editors. Each script controls AlphaSpell through its ARexx port and provides you with a powerful GUI for correcting the misspellings in your document. The GUI will also guess at the correct spellings of words for you, and it will add words to a user dictionary.

AlphaSpell is a fast and powerful spell checking utility. It uses a compressed dictionary format that it can read faster than a regular text file. It will also let you build and maintain dictionaries in its special format, and it is very easy to create a dictionary with AlphaSpell. Just send it a sorted wordlist, which it will compress into its native dictionary format. If you don't have a wordlist, AlphaSpell can help you build one from text files. It will tally up how many times words appear in several documents and then build a list of the most common words for you. But you might not even need to use this feature. Several dictionaries are already available for AlphaSpell. Besides the English dictionary that comes with it, there are dictionaries available for several languages.

1.3 Features

Spell Checking

An entire document

Interactively as you type

Checking for misused words

Guessing words

Pattern matching

Dictionary maintenance

Counting words

Building lists of words

Listing Anagrams

Arexx Port

Dictionary Formats

1.4 Spell Checking

Spell checking is the principal thing that AlphaSpell does ↩
. It can
quickly spell check an entire file, and with a text editor that is
customizable enough, it can provide transparent interactive spell checking.
It's designed in such a way that it does both sorts of spell checking in a
flash. When it spell checks an entire document, it first sorts the words,
then checks them against the dictionaries in order, so that it can begin
each new search without covering old territory. By quickly jumping ahead to
the words beginning with the first letter of the word it is searching for,
and by taking advantage of the dictionary layout to quickly skim past words
that don't match, AlphaSpell can check for a word in the blink of an eye.
This great speed makes it suitable for interactive spell checking.

SEE ALSO:

The CHECK command

The SEARCH command

Spell checking a document

Interactive Spell Checking

1.5 Spell checking a document

To spell check a document with AlphaSpell, use the CHECK ↩
command.
It will list any words it doesn't find. You can then search for these words in the document and correct those that need correcting.

It's best to do spell checking with the help of a GUI. Dirk Holtwick's MUISpell is a standalone GUI for spell checking documents, and my AlphaSpell GUI lets you spell check documents with a variety of different text editors.

SEE ALSO:
The CHECK command

1.6 Interactive Spell Checking

AlphaSpell can check words so quickly, you can use ↩
it for
interactive spell checking without slowing down your typing. What is required is a text editor that will let you remap the space bar to run AlphaSpell on the last word you typed. An editor with ARexx and the ability to remap all the keys should be sufficient for this. I have begun to use AlphaSpell for interactive spell checking with XDME. Let me explain how interactive spell checking works with XDME. Even if you don't use XDME, this explanation can help you adapt another text editor for interactive spell checking with AlphaSpell.

I have mapped the space bar to read the last word typed, which it then sends to AlphaSpell with the SEARCH command. What is usually sent to AlphaSpell will just be a word. But sometimes it will be a string with punctuation or special characters. So when AlphaSpell receives the string, it removes extraneous characters from both ends, so that the first letter is always an alphanumeric, and the last character is always a letter. When it is finished checking for the word, it writes the word to the environment variable word, and it puts a "1" or a "0" in the environment variable found. I use "0" and "1" so that XDME can use this value in a conditional. A "1" indicates that it found the word, and a "0" indicates that it did not. When a word isn't found, XDME calls an ARexx script that produces a display beep, and it indicates in the titlebar what word wasn't found. To speed up the interactive spell checking, XDME copies the dictionaries and AlphaSpell to the RAM disk. When you turn interactive spell checking off, it deletes these from RAM.

SEE ALSO:
The SEARCH command

1.7 Finding commonly misused words with AlphaSpell

Their are just sum errors spell checking won't catch. Four ↩
example,

homonyms, words witch sound alike, will knot be picked bye a spelling checker. But AlphaSpell can still help you with such words. If you create a list of words you are prone to misuse, you can use the COMMON switch with the CHECK command to check a document for them. This works just like spell checking, except that it lists the words it does find rather than the ones it doesn't.

SEE ALSO:

The COMMON option

1.8 Guessing words

To have AlphaSpell guess the correct spelling of a word, use the MATCH command.

AlphaSpell uses two different methods for guessing words. The default method is to use an algorithm based on the SoundEx algorithm. What this does is transform each string into a code that sort of represents the way it sounds. I say "sort of," because this algorithm is designed for use with English words, and the same letters often make different sounds in different English words. For example, "gh" can make a hard G sound, as in ghost, an F sound, as in rough, or no sound at all, as in through. This method prints a word whenever its code matches the code for the word given by the WORD option. Since the SoundEx algorithm is based on the way letters sound in English, it may be less effective for other languages. It is most useful when you make spelling mistakes because you tend to spell words as they sound to you. It is not quite as effective at helping you fix typos.

The second method is much more reliable, and it is just as reliable for other languages as it is for English. This method checks whether the edit distance between two words exceeds a certain value. You specify this value with the ED option. The edit distance between two words is the least number of deletions, insertions, and transpositions it takes to change one word into another. This value will be zero when the two words are the same, and in the worst case, when the two words share no letters in common, it will be the sum of their lengths.

I recommend a low value for the ED option when you want AlphaSpell to guess a short word, and a larger value when you want AlphaSpell to guess a longer word. For short words, large values will often list too many words, and for long words, small values may yield no words.

In the unregistered version, i.e. without the keyfile, you can set the edit distance no higher than two.

SEE ALSO:

The MATCH command

The ED=DISTANCE option

On the Edit Distance Algorithm

1.9 On the Edit Distance Algorithm

I came across the edit distance method in a book by Graham A
Stephen called _String Searching Algorithms_. I initially copied the
Wagner-Fischer algorithm for calculating the Levenshtein distance between
strings, which I then modified to handle transpositions and to calculate
the edit distance instead. The run time of this algorithm was O(mn), where
m and n are the string's lengths. In English, the run time was the product
of the string's lengths. I then reworked it until I got an algorithm whose
run time was approximately O(k), where k is the maximum edit distance
between words. I also decreased the time it took to complete each major
step. The original algorithm created a table and calculated each value of
the table by calculating four different values and taking their minimum. I
changed it to determine which value would be the minimum before actually
calculating any of the values. My algorithm takes only O(k), because it
doesn't reconstruct the table for each word. It uses what it can from the
last table created, and it stops when it can determine that the edit
distance will be greater than k.

SEE ALSO:
The MATCH command

Guessing words

The Levenshtein Distance

1.10 The Levenshtein distance

The Levenshtein distance between two strings is the minimum number
of insertions, deletions, and substitutions it would take to transform one
into the other. This differs from the edit distance by counting
substitutions the same as insertions and deletions. For the edit distance,
substitutions have a weight of two, because a substitution is equivalent to
an insertion and a deletion. But for the Levenshtein distance,
substitutions have a weight of one. I chose to use the edit distance rather
than the Levenshtein distance, because I didn't want words such as "cat"
and "dog" to have a distance of only three. That would cause AlphaSpell to
guess too many words. Both measures can be modified to handle
transpositions, as I did for the edit distance.

SEE ALSO:
The MATCH command

Guessing words

On the Edit Distance Algorithm

1.11 Pattern Matching with AlphaSpell

AlphaSpell no longer uses SH style pattern matching. It uses standard AmigaDOS pattern matching. It can do both case sensitive and case insensitive pattern matching. Case insensitive matching is the default. Set the CASE switch for case sensitive matching. Pattern matching is useful for guessing the correct spelling of a word, and it is also useful for cheating at crossword puzzles.

SEE ALSO:
The MATCH Command

1.12 Dictionary Maintenance

AlphaSpell offers a variety of features for creating and maintaining dictionaries. AlphaSpell lets you manipulate dictionaries like sets. It will combine two dictionaries into their union, their intersection, the asymmetric difference between the two, or the symmetric difference between the two. It will also add words to a dictionary from an unordered list.

SEE ALSO:
The MERGE command

1.13 Counting words

AlphaSpell will count the words in a file or dictionary according to the same rules it uses to recognize words when extracting them for spell checking. It recognizes as a word any string of two or more alphanumerics or apostrophes, which ends with a letter. AlphaSpell recognizes all the letters in the ISO character set.

SEE ALSO:
The COUNT command

1.14 Building clean lists of real words from large documents

Since Random House and Webster haven't made freely available word lists, and since it becomes tedious and time consuming to type in words from a dictionary, another method is needed for generating dictionary files. The best method available is to scan many files and count how many times each word in each file appears. The idea is that words which appear most frequently are correctly spelled words.

AlphaSpell has two commands to help you create dictionaries by this method. The first of these is the TALLY command. This command reads files and creates a frequency list. A frequency list has a word, a tab, and a decimal number on each line. The number indicates how many times the word was found. Because frequency lists can get huge, I added the LISTS option to the TALLY command for merging frequency lists together. Before merging files together, it is best to randomize the lines in them. This will make reading them much faster, as in minutes verses hours. I have written a separate program for this. Finally, the WEED command weeds through a frequency list, printing each word that has appeared at least as many times as you specify with the FREQ option. For this, it is important to use an ordered frequency list, just as it is created by AlphaSpell.

TECHNICAL DETAILS

The TALLY command loads words into a binary tree, and that takes a long long time for large lists of already alphabetized words, but much less time for unordered lists. The TALLY command gives ordered lists as output. The WEED command just prints words as it finds them and does not load them into a tree. So it is just as fast with ordered or unordered lists. It will generate ordered output only if it receives ordered input.

SEE ALSO:
The TALLY command

The WEED command

1.15 Listing anagrams

An anagram is a word with exactly the same letters as another word.
With the ANAGRAMS option for the MATCH command, AlphaSpell will search for all the anagrams of a word.

SEE ALSO:
The ANAGRAMS option

1.16 AlphaSpell's ARexx Port

With version 6, AlphaSpell now has an ARexx port. To use the port,
run AlphaSpell with the AREXX command. To close down the port, pass the QUIT command to the port. AlphaSpell's port name is ALPHASPELL.

SEE ALSO:
The AREXX command

1.17 Dictionary Formats

I have dubbed AlphaSpell's native dictionary format ASP. I introduced this format in AlphaSpell V. It is a compressed format that is usually about 60maller than the text file it is compressed from. The compression makes it faster to read, and some of AlphaSpell's algorithms take advantage of special features of the ASP format for even greater speed. (NOTE: The unregistered version of AlphaSpell will not compress ASP dictionaries. It will write uncompressed ASP dictionaries.)

The ASP format is similar to the SPL format used by the MS-DOS spelling checker amspell. Although the SPL is invariably larger than the ASP format, I mistakenly thought that it might have some speed advantages for ASP. So I implemented routines for reading, writing, and searching SPL dictionaries. As it turned out, the ASP format was considerably faster, because it used buffered IO. So I modified the SPL routines to also use buffered IO. This introduced a bug into the search routine, and as it turned out, SPL was still slower. So I scrapped everything except the ability to read SPL dictionaries. I left this in in case you ever come across an SPL dictionary and would like to convert it to the ASP format.

Finally, AlphaSpell can read and write uncompressed ASCII word lists.

When AlphaSpell is matching or spell checking, it requires ASP dictionaries. This is because some of the algorithms make use of special features of this format. Besides that, there is no gain to be had from being able to read or check through SPL dictionaries or text files. ASP is the best format to use, and you can easily convert SPL and ASCII files into ASP format. AlphaSpell recognizes ASP dictionaries by the suffixes ".ald" and ".amd". The first identifies a lowercase dictionary, and the second identifies a mixed case dictionary.

AlphaSpell recognizes both lowercase and mixed case dictionaries, so that you can do case insensitive spell checking on ordinary words and case sensitive spell checking on names, acronyms, and the like. Ordinary words should go into lowercase dictionaries. Names and acronyms should go into mixed case dictionaries. That way, it can recognize that "mary" is a misspelling while "Mary" is not, and it will recognize that "marry" and "Marry" are both correct spellings.

To convert a text file or an SPL dictionary to an ASP dictionary, use the DUMP command. Give the appropriate suffix to the output file. It will read a file as an SPL dictionary if its suffix is ".spl". It will read it as an ASP dictionary if you give it the appropriate suffix, and it will read anything else as a text file.

1.18 Usage of AlphaSpell

With AlphaSpell VI, AlphaSpell uses ReadArgs, which is the ↩ standard among Amiga programs. It has several commands, each with its own template. These are:

```
 ADD

 WORD
/K,
 PATH
/K,
 FROM
/M,
 TO
/A


 AREXX

ARGS
/F


 CHECK

 FROM
/A,
 TO
/A,
 PATH
/K,
 IN
/M,
 COMMON
/S,
 TEX
/S


 DUMP
    FROM/A,TO/A


 COUNT

 FROM
/A,
 BASE
/N/K,
DICT
/S,
 TEX
/S


 MATCH

 WORD
/A,
 TO
```

```
/A,
 PATH
/K,
 IN
/M/A,
 CASE
/S,

 ANAGRAMS
/S,
 ED=DISTANCE
/K/N


 MERGE

 FIRST
/A,
 SECOND
/A,
 TO
/K,
 AND=INTERSECTION
/S,

 SUB=DIFFERENCE
/S,
 OR=UNION
/S,
 XOR
/S


 SEARCH

 FOR
/A,
 PATH
/K,
 IN
/M/A,
 TEX
/S


 TALLY

 FROM
/M/A,
 TO
/A,
 PATH
/K,
 LISTS
/S,
 TEX
/S
```

```
VERSION

WEED

FROM
/A,
TO
/A,
FREQ
/K/N
```

1.19 The TEX option

Use this option to have AlphaSpell discard TeX commands when it is scanning a file (CHECK, COUNT, TALLY) or a word (SEARCH).

1.20 The DUMP command

```
DUMP FROM/A,TO/A
```

The DUMP command dumps the contents of a dictionary into a new dictionary. This command is useful when you want to convert a dictionary from one format to another. Its output is the same as MATCH #?, but it's faster, because it does no pattern matching.

1.21 The AREXX command

```
        AREXX
ARGS
/F
```

Use this command to have AlphaSpell open an ARexx port and serve as an ARexx command host. AlphaSpell's ARexx commands include all the commands you can use from the shell (except for AREXX) plus some others. These are:

```
ABOUT

ANNOUNCE
MESSAGE/F

FLUSH

QUIT

REPSTR
   STRING/A,SEARCH/A,REPLACE/A,TIMES/N
```

```
                  SETBUFFER
                     SIZE/N/A

                  WHO
                 EXAMPLE:

    AlphaSpell AREXX

SEE ALSO:
                  AlphaSpell's Commands

                  The ARexx port
```

1.22 The ARGS option

When you use the AREXX command, you can include the name of an ARexx script along with any arguments you want to pass to the script. The script will start out with ALPHASPELL as the default ARexx port. AlphaSpell will recognize scripts with the extension ".asp" as AlphaSpell scripts.

1.23 The WHO command

WHO

This command checks for the keyfile, and if a keyfile exists, it stores in RESULT the name of the person the keyfile belongs to. If there isn't a keyfile, or if there is a bogus keyfile, it stores "No One" in RESULT. This command is useful for putting the user name into requesters, and it is useful for checking whether there is a valid keyfile.

1.24 The ABOUT command

This command displays an Intuition requester with information ABOUT AlphaSpell.

1.25 The FLUSH command

FLUSH

This command frees up memory that AlphaSpell has previously used and no longer needs. It is useful when a script repeatedly executes commands, such as TALLY and ADD, that eat up a lot of memory. [Technical note: AlphaSpell regularly allocates memory with AllocRemember(), and the FLUSH command calls FreeRemember().]

1.26 The REPSTR command

```
        REPSTR    STRING/A,SEARCH/A,REPLACE/A,TIMES/N
```

This command replaces substrings within a string with other substrings and writes the modified string in RESULT. STRING specifies the string to change, SEARCH specifies the string to search for, and REPLACE specifies the string to replace the search string with. If you pass a value for TIMES, it replaces the search string only that many times. If you pass no value to TIMES, it replaces every instance of the search string.

This command is useful when you need to modify strings because of some idiosyncrasy of your text editor. For example, XDME regards apostrophes as special characters. So when I pass XDME a string to search for, I make sure that any apostrophes in it are escaped.

EXAMPLE:

```
    ADDRESS ALPHASPELL.1 REPSTR "can't" "'" "\'"
    SAY RESULT
```

1.27 The QUIT Command

Use QUIT to close down AlphaSpell's ARexx port and to exit from AlphaSpell.

1.28 The SETBUFFER command

```
        SETBUFFER   SIZE/N/A
```

Use SETBUFFER to set the size of the buffer that AlphaSpell uses when it reads dictionaries. Up to a certain point, a bigger buffer means faster access to the dictionaries. If you set a size greater than available memory will allow, it just makes the buffer as large as memory will allow.

1.29 The ANNOUNCE command

```
        ANNOUNCE    MESSAGE/F
```

Use ANNOUNCE to display a message to the user with an Intuition requester. You can include linefeeds in a message with \n.

1.30 The WEED command

```
            WEED
 FROM
/A,
 TO
/A,
 FREQ
/A/N
```

Use WEED to generate a wordlist or dictionary out of a frequency list created with the TALLY command. It will write out each word that appeared at least as many times as the word FREQuency you set. The higher you set FREQ, the greater your chances are of getting only correctly spelled words.

EXAMPLE:

```
AlphaSpell WEED FROM freqlist TO common FREQ 5000
```

This reads freqlist, a list of word frequencies, and creates a list of all the words that appeared at 5000 times.

SEE ALSO:

Building clean lists of real words from large documents

1.31 The FROM option for the WEED command

Use FROM to indicate the name of the frequency list you want the WEED command to generate a dictionary or wordlist from.

1.32 The FREQ option

Use FREQ to set the minimum word frequency for the WEED command. It will print each word that appeared at least that many times.

1.33 The VERSION command

```
VERSION
```

Use VERSION to make AlphaSpell spit out the current version and revision of AlphaSpell. When executed from the ARexx port, it will also store the version number in RESULT. This is useful for making sure that an ARexx script is using an up to date version.

1.34 The TALLY command

```
            TALLY
 FROM
/M/A,
 TO
/K,
 PATH
/K,
 LISTS
/S,
 TEX
/S
```

Use this command to generate a list of the word frequencies in a set of files. It will list each word that appeared in any of the files, followed by the total number of times it appeared. Together with the WEED command, the TALLY command is useful for creating useful dictionaries out of electronic documents you may find on the Internet or elsewhere. High frequencies are a good indicator that a string is a correctly spelled word. With the LISTS switch set, AlphaSpell will read frequency lists and merge them together into one list. The default is to read regular documents.

EXAMPLE:

AlphaSpell TALLY *.txt TO freqlist

This builds a list of word frequencies from all the files matching the *.txt wildcard pattern.

SEE ALSO:
Building clean lists of real words from large documents

1.35 The FROM option for the TALLY command

Use FROM to indicate which files you want the TALLY command to tally up the words in.

1.36 The LISTS switch

```
Use this switch to make the TALLY command read frequency  ↩
   lists
```
instead of regular files. It will then combine together the totals from each list.

EXAMPLES:

AlphaSpell TALLY #? TO freqlist LISTS

This reads all the files in the current directory, which should all be frequency lists, and merges them into one list called "freqlist."

```
    AlphaSpell TALLY TO masterlist masterlist words LISTS

    This adds the frequencies listed in the file "words" to those in the
master frequency list "masterlist."

SEE ALSO:
                Building clean lists of real words from large documents
```

1.37 The MERGE command

```
                MERGE
 FIRST
/A,
 SECOND
/A,
 TO
/K,
 AND=INTERSECTION
/S,

 SUB=DIFFERENCE
/S,
 OR=UNION
/S,
 XOR
/S
```

Use this command to combine two wordlists or dictionaries together into a new one. You can take the UNION, the INTERSECTION, the asymmetric DIFFERENCE of the two, or the symmetric difference of the two. The default is the UNION of the two. This command is mainly for the purpose of adding the words in a user dictionary to the main dictionary. It is also useful for creating and maintaining dictionaries.

1.38 The FIRST option

The MERGE command reads two files. Use FIRST to name the first one.

1.39 The SECOND option

The MERGE command reads two files. Use SECOND to name the second one.

1.40 The AND=INTERSECTION option

Use AND or INTERSECTION to generate the intersection of ↩
the words
in two dictionaries or wordlists. This is the listing of all the words they share in common. To put it another way, it is the listing of each word that is in both the first AND the second file that AlphaSpell reads.

EXAMPLE:

AlphaSpell ukacd.ald AND common.ald TO common.ald

This compares a list of common words, perhaps generated by tabulating word frequencies in files, with the UK Advanced Cryptics dictionary. The effect is to remove from common.ald any words not found in the much larger dictionary.

SEE ALSO:
Dictionary maintenance

1.41 The OR=UNION option

Use OR or UNION to generate the union of the words in two dictionaries or wordlists. This is the listing of all the words in both. To put it another way, it is the listing of each word that is in either the first OR the second file that AlphaSpell reads.

EXAMPLE:

AlphaSpell MERGE English.ald OR User.ald TO English.ald

This adds the words in the lowercase user dictionary to the main lowercase dictionary.

SEE ALSO:
Dictionary maintenance

1.42 The XOR option

Use XOR to generate the list of words that are in either the first or second file that AlphaSpell reads but not in both. XOR stands for eXclusive OR, and it is the opposite of AND.

1.43 The SUB option

Use SUB or DIFFERENCE to generate a list of all the words ↩
that are
in the first file AlphaSpell reads but not in the second. This is in effect a SUBtraction of one file from the other.

```
EXAMPLE:

    AlphaSpell Spangalese Spangalese.amd SUB TO Spangalese.ald

    This creates a lowercase Spangalese dictionary by subtracting the mixed
case words in a Spangalese wordlist.

SEE ALSO:
                Dictionary maintenance
```

1.44 The MATCH command

```
                MATCH
 WORD
/A,
 TO
/A,
 PATH
/K,
 IN
/M/A,
 CASE
/S,

 ANAGRAMS
/S,
 ED=DISTANCE
/K/N
```

Use the MATCH command to generate a list of words that in some way match a particular WORD or wildcard pattern. If you pass a wildcard pattern, it does case insensitive pattern matching by default. You can choose case sensitive pattern matching instead by setting the CASE switch. If you pass a WORD and don't set any special options, it does SoundEx style matching. If you set the ANAGRAMS switch, it matches a WORD with its anagrams. Finally, if you specify an Edit Distance with the ED option and don't set the other switches, it will match words close in spelling.

The most useful options for guessing the correct spelling of misspelled words are SoundEx matching and Edit Distance matching. SoundEx matching is useful for matching a word with words that sound sort of like it, and Edit Distance matching is useful for correcting typos. Edit Distance matching is limited to an edit distance of 2 for unregistered users.

```
EXAMPLES:

    AlphaSpell MATCH p*t PATH $DDIR $Dict

    This lists all the words in the dictionaries beginning with the letter
p and ending with a t, such as pot, Pat, and proletariat. It uses the
environment variables created by the Install script. $DDIR holds the name
```

of the dictionary drawer, and $Dict holds the names of the dictionaries.

AlphaSpell MATCH WORD ??* IN Spangalese TO Spangalese

This removes solitary letters from a list of Spangalese words.

AlphaSpell MATCH ???? PATH $DDIR *.ald

This searches the lowercase dictionaries in the dictionary drawer for all four letter words.

AlphaSpell MATCH *[A-ZÀ-Þ]* TO Spangalese.amd Spangalese CASE

This creates a dictionary of mixed case Spangalese words from a Spangalese wordlist.

SEE ALSO:

Guessing Words

Pattern Matching

1.45 The WORD option for the MATCH command

Use WORD to specify the word or wildcard pattern that you want to match against the dictionaries.

1.46 The CASE switch

Use CASE to indicate that you want to do case sensitive pattern matching. This switch has an effect only if you pass a wildcard pattern to the MATCH command. This may change in the future, as I may make available both case sensitive and case insensitive versions of some of the other matching routines. For the moment, Anagram matching and Edit Distance matching are both case sensitive, and the others are case insensitive.

1.47 SoundEx Matching

SoundEx matching is the default for the match command. This works by generating a hash code for a word which roughly indicates how it sounds. AlphaSpell then generates hash codes for each word in the dictionary and compares them with the target's hash code. This method is most useful when you spell words by the way they sound to you.

EXAMPLES:

AlphaSpell MATCH ghoti Data:Dictionaries/*.ald TO matches

This uses the SoundEx method to search for words that might sound like ghoti, and it will list words such as goat and goatee, putting them in the

file called "matches." It is not sophisticated enough to know that ghoti is
a homonym for fish: touGH + wOmen + cauTIon.

1.48 The ANAGRAMS option

Use this option to match the anagrams of a word. An anagram is a
word with exactly the same letters as another word. For example, cats,
acts, and scat are anagrams. This option is a holdover from the days when
AlphaSpell used more primitive guessing routines. It remains because it is
useful for cheating at word games in the newspaper.

SEE ALSO:
Listing anagrams

1.49 The ED=DISTANCE option

This option lists each word within a specified edit distance from
the target WORD. The edit distance between two words is the minimum number
of deletions, insertions, and transpositions that it takes to transform one
word into the other. Without the keyfile, which you get when you register,
you can only specify an edit distance of 1 or 2. This is fine for short
words, but setting the edit distance higher is often useful for matching
longer words. This is the most sophisticated, and it is probably the most
useful, method for guessing the correct spelling of misspelled words.

EXAMPLES:

```
AlphaSpell MATCH ED 2 WORD lase PATH Data:Dictionaries/ *.ald *.amd TO matches
```

This checks the word "lase" against the dictionaries in
Data:Dictionaries named by the wildcard patterns. It writes to "matches"
any words with an edit distance of two or less from "lase," words such as
"lace," "laser," and "case."

1.50 The CHECK Command

```
CHECK
 FROM
/A,
 TO
/A,
 PATH
/K,
 IN
/M,
 COMMON
/S,
 TEX
```

```
                /S

        This command checks the spelling of words in a document and returns
a list of all the words it didn't find. If the COMMON switch is set, it
returns a list of all the words it did find instead. It sends its output to
the file specified by TO or else to standard output. It can read any number
of dictionaries. Just name them. If you specify a PATH, it will look for
the dictionaries there.

EXAMPLE:

    AlphaSpell CHECK FROM letter.txt TO unfound PATH $DDIR $Dict

    This spell checks "letter.txt" and lists unfound words in the file
"unfound." It uses the environment variables $DDIR and $Dict to name the
dictionary drawer and the dictionaries.

    AlphaSpell CHECK FROM T:temp TO T:temp PATH $DDIR *.ald *.amd

    This spell checks a temporary file saved by a text editor and
overwrites it with a list of unfound words. This is an economical use of
the same file when the actual text is held in the buffer of a text editor.
It uses $DDIR to name the dictionary directory and wildcard patterns to
read all the dictionaries in the directory.

SEE ALSO:
                Spell Checking

                Spell Checking a Document
```

1.51 FROM in the CHECK Command

```
        Use FROM to specify the file you want to spell check.
```

1.52 The TO option in general

```
                        Use TO to specify AlphaSpell's output file. The extension ↩
                           you give
to the file will determine the format that AlphaSpell writes it in.
AlphaSpell recognizes the following two file extensions:

    .ALD    AlphaSpell format Lowercase Dictionary
    .AMD    AlphaSpell format Mixedcase Dictionary

        If the file name has either of these two extensions, it writes it
as an ASP dictionary. Otherwise, it writes a straightforward text file.
Whether you choose a lowercase or mixed case dictionary as output, that
doesn't affect the output. It makes a difference only when AlphaSpell uses
the dictionary for spell checking. It is up to you to make sure that only
mixed case words go into mixed case dictionaries and only lowercase words
go into lowercase dictionaries.
```

SEE ALSO:
Dictionary Formats

1.53 The PATH option

Use PATH to specify the name of the directory that AlphaSpell should search for multiple files in, typically dictionaries. Some commands have an IN/M or a FROM/M option. If the PATH is set, it searches for these files there. Whenever you include a complete path name as part of a file name (or file name pattern), AlphaSpell ignores the setting of the PATH option. For example,

AlphaSpell TALLY PATH Work:Texts/ *.txt Work:Stuff/*.txt *.doc

will read Work:Texts/*.txt, Work:Texts/*.doc, and Work:Stuff/*.txt.

1.54 The IN option

Use IN to specify the dictionaries AlphaSpell should read with the CHECK, MATCH, and SEARCH commands. You can use the PATH option to specify where AlphaSpell will look for the dictionaries you name with this option.

1.55 The COMMON Switch for the CHECK Command

Use the COMMON switch to have the CHECK command write each ↩
word
that it does find instead of each word that it doesn't find. This is useful when you want to check a document for commonly misused words.

EXAMPLE:

AlphaSpell CHECK FROM letter.txt COMMON Work:Dictionaries/confusing.ald

This checks whether letter.txt contains any of the words in confusing.ald, and it lists any it finds.

SEE ALSO:
Finding commonly misused words with AlphaSpell

1.56 The COUNT Command

```
        COUNT
 FROM
/A,
 BASE
/N/K,
 DICT/S
,
 TEX
/S
```

This command counts the words in a file and writes the result to an environment variable called words. It writes its result in the specified base, which can be anything from base 2 to base 36. The default is base 17. Just kidding. The default is base 10. Set the DICT switch if you want to count the words in a dictionary instead of in a text file.

EXAMPLE:

```
AlphaSpell COUNT cats.txt BASE 16
```

This counts the words in cats.txt and stores the total in the environment variable words in hexadecimal.

1.57 The DICT switch

Use this switch to COUNT the words in a dictionary.

1.58 The FROM option in the COUNT Command

Use FROM to name the file you want to count the words in.

1.59 The BASE option in the COUNT Command

This is here only because it was easy to include. Use BASE to specify the base you want AlphaSpell to write the total in.

1.60 The SEARCH command

```
        SEARCH
 FOR
/A,
 PATH
/K,
 IN
/M/A,
 TEX
```

```
                /S
```

Use this command to check for a single word in the named dictionaries. This command was originally written for debugging the spell checking routines, but it is now designed for interactive spell checking. To use it for interactive spell checking, set up your text editor to call AlphaSpell every time you finish typing a word. AlphaSpell will indicate whether it found the word by writing to an environment variable called found. It will write a "1" there if the word was found and a "0" otherwise. Since the word you type might be adjacent to punctuation, AlphaSpell parses the string you typed into a word, spell checks for the word, and writes the word it searched for into an environment variable called word.

EXAMPLE:

```
    AlphaSpell SEARCH FOR bat PATH $DDIR English.??[!x] User.??[!x] common.??[!x]
```

This checks "bat" against the dictionaries in the dictionary directory, checking a short dictionary of common words first. (It reads files in the order last to first). If it doesn't find the word there, it checks the user dictionaries, and it checks the main dictionary only if it still hasn't found the word. By checking shorter dictionaries of more commonly used words first, it takes less time to search for the word.

SEE ALSO:

Spell Checking

Interactive Spell Checking

1.61 The FOR option

Use FOR to specify which word you want to SEARCH FOR.

1.62 The ADD Command

```
                ADD
 WORD
/K,
 PATH
/K,
 FROM
/M/A,
 TO
/A
```

This command is designed for adding words to a user dictionary. It can also be used for adding words to the main dictionary, but it will take too long. You should use the MERGE command for adding words to the main dictionary. This command will have AlphaSpell read FROM each of the named input files, and then it will read the file that it will write TO. All of these files should be either wordlists with one word to a line or

dictionaries. It will then write a new dictionary made up of all the words
from the files it read. It will also add a single WORD if you specify one.

EXAMPLES:

    AlphaSpell ADD WORD McDuff TO Names.amd

    This adds the name "McDuff" to a mixed case dictionary containing
names.

    AlphaSpell ADD words TO User.ald

    This adds the words in the wordlist "words" to a user dictionary.

1.63 The WORD option for the ADD command

        Use WORD to specify a single word that you want to add to a
dictionary or wordlist. This option may be used in conjunction with FROM.

1.64 The FROM option in the ADD Command

        Use FROM to indicate which dictionaries or lists of words you want
to add to another dictionary or list of words. You can name as many as you
want. The file you are adding words to will also be read. So you don't have
to name it with this option. If you do, no harm will be done, but it will
take slightly longer, because it will read the same file twice.

1.65 The TO option in the ADD command

                        Use TO to name the file you want to add words to. Since ↩
                           AlphaSpell
is adding words to the file, it will read the output file before it writes
to it. If you use redirection operators instead of the TO option,
AlphaSpell will overwrite its output file without reading it. So make sure
you use the TO option if you want to add words to a file. The TO option is
also essential if you are writing a dictionary.

SEE ALSO:
                The TO option in general

1.66 Legal Matters

                Copyright

1.67 Copyright and Trademark

AlphaSpell Copyright © 1992-6 Fergus Duniho

You are NOT ALLOWED to modify AlphaSpell VI or the documentation in any way. Packing and archiving do not count as modifications. You are NOT allowed to decompile AlphaSpell. This copyright does not extend to the words in the dictionaries. You may use them in any way you please, as Humpty Dumpty advocates, though others have questioned the wisdom of this approach.

These restrictions do not apply to AlphaSpell IV, which was copylefted under the GPL. Beginning with version 5, AlphaSpell is not under the GPL; It is shareware.

The name "AlphaSpell" is a trademark used by Fergus Duniho.

1.68 Distribution

The unregistered version of AlphaSpell is freely distributable by any normal electronic means. Anyone may distribute it so long as they keep the entire contents of this package intact and unchanged. This package contains the following: AlphaSpell, AlphaSpell.guide, AlphaSpell.gui, Dict.ald, Dict.ldx, Dict.amd, Dict.mdx, Install, and Register.

AlphaSpell may NOT be distributed by anyone whose advertising is likely to mislead people into believing that AlphaSpell is public domain. If anyone advertises that AlphaSpell is available on one of the disks they distribute, they must briefly explain what shareware is and make it clear that some of the disks they distribute contain shareware. Otherwise, they may not distribute AlphaSpell.

If you got AlphaSpell from a disk that you paid money for, you still haven't paid for AlphaSpell until you have paid me, Fergus Duniho, the registration fee. What you paid money for was the disk the program came on, not the program itself. If you decide to continue using AlphaSpell after a period of one month, you must pay me the registration fee.

AlphaSpell's keyfile may not be redistributed by anyone by any means. Redistributing it is software piracy.

1.69 Disclaimer

By using this product, you accept the FULL responsibility for any damage or loss that might occur through its use or the inability to use it. The author of this program can NOT be held responsible.

Furthermore, I do not guarantee that any AlphaSpell dictionary is complete or 100% accurate. I cannot be held liable for the inaccuracy or incompleteness of any AlphaSpell dictionary. I cannot even be held liable when an AlphaSpell dictionary is a gross misrepresentation of the language it is supposed to be a dictionary for. This is actually a real possibility, as I could mistranslate a character set used in a wordlist for a language I don't know. So be on your guard.

Moreover, I cannot be held liable for the presence of offensive words in any AlphaSpell dictionary.

1.70 Legal Use

You may use AlphaSpell free for a trial period of one month. If you decide to continue using it after that time, you must pay the shareware fee. This fee is $20.00 in United States currency or $30.00 in Canadian currency. An alternative to paying the shareware fee is to help me expand the market for AlphaSpell. You can do this by providing me with an AlphaSpell GUI for a text editor that doesn't yet have one, by providing me with a dictionary for a language AlphaSpell doesn't yet support, or by translating the documentation into another language.

1.71 Using the AlphaSpell GUI

The AlphaSpell GUI has three different windows.

The main window

The prefs window

The learn window
The GUI supports several editors.

How to add support for an unsupported editor.

A note to text editor authors

1.72 The main window

```
                    This window is designed to sit unobtrusively above the window ↩
                       that your
document is in. That way, you don't have to move it around to see the words
you want to check out and maybe change. This window lets you check your
document for misspellings by letting you search for and change the words
that AlphaSpell didn't find when it spell checked your document.

· The AlphaSpell GUI Copyright © 1995-6 Fergus Duniho
|
                     Find

                    Select
                     Learn
                     Prefs
                   |<
                   <<
                   >>
                   >|
                  |
                    Replace

                     Guess
                  @|
                   |
                   2
```

1.73 The Select Button

This button displays a listview of all the words that AlphaSpell didn't find when it spell checked your document. If you select a word from the listview, it shows up in both of the string gadgets, and the listview disappears. If you click on the close button of the listview, it goes away without changing anything.

1.74 The Learn Button

This button stores the word in the top string gadget to a list of words that you can save to your user dictionary when you're finished spell checking. It will put a lower case word in a list of lower case words, and it will put a mixed case word in a list of mixed case words. The Learn button is mainly for learning correct words that AlphaSpell didn't find in any of its dictionaries. But you can also add any word you want by entering it into the top string gadget before clicking on the Learn button.

1.75 The Find Button

This button searches through the text for the next (or first) occurrence of the word in the
string gadget
to its right. If that word has been newly changed, it searches for the first occurrence of the word. Otherwise, it searches for the next occurrence of the word.

1.76 The String Gadget for the Find String

This string gadget contains the string that you want to search ↩ through the text for. This will typically be one of the words that AlphaSpell didn't find, and you can move through the words in this list with the
|<
,
<<
,
>>
, and
>|
buttons. You can change the value of this gadget with the
Select
button or by typing in a new word. You can learn the word in this gadget with the
Learn
button.

1.77 The << Button

This button moves you backwards through the list of unfound words. If you're at the beginning of the list, it moves you to the end.

1.78 The >> Button

This button moves you forward through the list of unfound words. If you're at the end of the list, it moves you to the beginning.

1.79 The |< Button

This button moves you to the first word in the list of unfound words.

1.80 The >| Button

This button moves you to the last word in the list of unfound words.

1.81 The Guess Button

This button has AlphaSpell guess at what the word in the bottom string gadget is supposed to be. It displays its guesses in a listview. When you select a word from it, the listview goes away, and the word shows up in the bottom string gadget. If you click on the listview's close button, the listview goes away, and nothing is changed. How AlphaSpell guesses a word is determined by the two gadgets to its right: the cycle gadget and the slider gadget.

1.82 The Guessing Method Cycle Gadget

This gadget cycles between four options that affect guessing. These are: Edit Distance, SoundEx, Anagrams, and Case Sensitive. When you select the Guess button, AlphaSpell matches the word in the Replace string gadget with the words in the dictionaries, and it returns a list of the matches in a listview. The first option selects Edit Distance matching. When this option is set, the slider gadget to the right is activated, and the Guess button searches for words whose edit distance from the target word is less than or equal to the value set by the slider gadget. When the SoundEx option is set, it finds words phonetically similar to the target word. When the Anagrams option is set, it finds anagrams of the target word. When the Case Sensitive option is set, it will do case sensitive pattern matching when the target word is a wildcard pattern. Otherwise, it will do SoundEx matching.

1.83 The Edit Distance Slider Gadget

```
                    This gadget sets the
                maximum edit
                 distance used for edit distance
matching. The edit distance is the minimum number of deletions, insertions,
and transpositions it takes to change one word into another. Using ed(x,y)
to mean the edit distance between the words x and y, the following theorems
are true:

        ed(x,y) == 0 if and only if x == y
        ed(x,y) == ed(y,x)
        ed(x,y) <= ed(x,z) + ed(y,z)
```

1.84 The Replace Button

```
                    When you've found a word with the
                Find
                 button, you can change it to the
string in the
                lower string gadget
                 by clicking on this button.
```

1.85 The String Gadget for the Replace String

```
                    This string gadget is supposed to hold the string that you ↩
                       want to
replace the
                Find
                 string with. You can change the value of this string
gadget with the
                Guess
                 button or by typing in it. Also,
it will change to the Find string whenever you move through the list of
unfound words. This is to ease guessing of each unfound word, as well as to
easily let you edit any changes you want to it.
```

1.86 The Prefs Button

```
                    This button opens up the
                Preferences window
                .
```

1.87 The Preferences Window

In this window, you can select your preference settings.

These are:

Language

You may select the language you want to check for from a listview. The listview shows you the languages you currently have installed. To install more languages, create a directory for each new language in AlphaSpell:Dict/. The list of available languages is identical with the list of drawers in AlphaSpell:Dict/.

Dictionaries

A list or wildcard pattern indicating the dictionaries that AlphaSpell uses for spell checking and for guessing. It looks for the dictionaries in the drawer for the specified language. To make it easy to switch between languages, it's best to use simple wildcard patterns here.

If you want to use dictionaries for languages besides the selected language, you can do so by including the full path name.

User Dictionary

The stem name for your user dictionary. This is so it knows where to send words that you want to learn. If you want the user dictionary used for spell checking and for guessing, you have to include it along with the other dictionaries you list for those purposes.

To save your preferences for future use, click on the SAVE button. To use your preference settings temporarily, without changing them permanently, click on the USE button. To cancel any changes you make, click on the CANCEL button or the CLOSE gadget.

1.88 The Learn Window

When you click on the CLOSE gadget once you're all finished spell checking, the Learn window will pop up if you selected any words for learning. It will display mixed case words in one listview and lower case words in the other. Each listview has a REMOVE WORD button for removing words from it. The mixed case listview has a MOVE WORD button for moving words from it to the lower case listview. When a word is moved from the mixed case list, it will be converted to lowercase. When you're satisfied with the words you want to add to your user dictionary, click on the SAVE WORDS button. If you decide not to save any words, click on the CLOSE gadget.

1.89 How to adapt the AlphaSpell GUI script for other text editors

To adapt one of the ASpell ARexx scripts for your editor, all you need to change are five functions at the end of the script:

FindWord(),

ReplaceWord(),

SaveTemp(),

GetEditPort(),

GetScreen(). You should set EDITPORT to the name of your text editor's ARexx port. Is is easiest to adapt ASpell.ed, ASpell.elx, or ASpell.xdme. ASpell.ged has some extra code in it, which you would have to delete.

1.90 FindWord()

This function is used for finding a word within a document. The first thing it should do is read the value of the target gadget. It does this with the lines:

```
read target
wrd = RESULT
```

These lines should be the same no matter what editor you use. The variable wrd holds the string that you want your editor to search for.

The next thing to do is check the value of arg(1). If its value is 0, FindWord should search for the first occurrence of wrd within your document. You can have FindWord do this by moving to the top before it begins its search, or you can use a find instruction that can be instructed to search for the first word. FindWord() should return 1. Here's an example using pseudo code:

```
ADDRESS
IF arg(1) = 0 THEN DO
    "FIND" wrd "CASE WHOLE FIRST"
END
ELSE DO
    "FIND" wrd "CASE WHOLE"
END
ADDRESS
RETURN 1
```

Notice that the searching done here is case sensitive. This just

makes things easier. Also note that it does whole word searching. This is because you are always searching for whole words, and you don't want it to stop every time it finds a small word as a substring of a larger word. If your editor lacks a whole word search mode, here's another example:

```
ADDRESS
IF arg(1) = 0 THEN DO
    TOP
    FIRST
END

DO FOREVER
    "FIND" wrd "CASE"
    READ fail
    IF RESULT > 0 THEN DO
        TOP
        FIRST
        LEAVE
    END
    READ currentword
    cword = RESULT
    IF WordComp(cword, wrd, 1) = 1 THEN LEAVE
END
ADDRESS
RETURN 1
```

This example uses a command for moving to the top of the document (TOP), a command for moving the beginning of a line (FIRST), and a command for reading information from the program (READ).

The variable fail in this example contains a nonzero value if the last command failed, or a 0 if it didn't. This is to check whether the FIND command found another word. The FindWord() in ASpell.xdme works something like this. If you can't tell whether a command has failed, you can check whether the cursor has moved by comparing the old and new positions of the cursor. This is what FindWord() in ASpell.ed does.

The variable currentword contains the current word in the document. This word is compared to the word you're searching for, using the WordComp() function, which is already included in the script. The third argument in WordComp tells it to start comparing at the beginning of cword. This is required when the first argument passed to WordComp is a single word. For a real working example of this method, take a look at FindWord() in ASpell.xdme.

You can also give WordComp a whole line for its first argument. If you do, you need to pass the X position of the cursor as the third argument. For a real working example of this method, take a look at FindWord() in ASpell.ed.

1.91 ReplaceWord()

```
This function replaces the current word with the word in ↩
   the
```

```
replacement string gadget. It begins by reading the value of this gadget:

read replacement
newword = RESULT

        Depending on how your replace command works, you may need to read
the value of the target gadget. The
                pseudo code
                 examples here require it.
Take a look at ASpell.xdme and ASpell.elx for examples that don't.

read target
wrd = RESULT

        Here's an example using the REPLACE command:

ADDRESS
"REPLACE" wrd newword
ADDRESS
RETURN

        Here's an example using DELCHAR and INSERT:

ADDRESS
"DELCHAR" Length(wrd)
"INSERT" newword
ADDRESS
RETURN
```

1.92 SaveTemp()

```
                        This function saves the current document as a temporary ↩
                           file. It
uses the file name that the script already stored in the variable tempfile.
Here's an example in
                pseudo code
                :

ADDRESS
"SAVEFILE TO" tempfile
ADDRESS

        It is important that your SaveTemp() function does not change the
name of your document. With some text editors, you may have to read the
file name, save the file, then change the file name back. Here's an
example:

ADDRESS
READ filename
oldname = RESULT
"SAVEFILE TO" tempfile
"CHANGENAME" oldname
ADDRESS
```

Take a look at ASpell.ed and ASpell.ged for real working examples that do this.

1.93 GetEditPort()

This function returns the name of the ARexx port to use for communicating with the text editor. If your editor doesn't use a named port, it should return the empty string. Generally, it checks whether the current port belongs to your editor. If it doesn't belong to it, it checks whether a port to your text editor is open. If it is, it returns the name of that port. If not, it exits from the script. Here's an example:

```
GetEditPort:
IF Abbrev(Address(), "FREXXED.") = 1 THEN RETURN Address()
IF ~SHOWLIST("P", "FREXXED.1") THEN DO
    CALL rtezrequest "FREXXED.1 unavailable", "_Abort", "Missing Port:"
    EXIT
END
RETURN "FREXXED.1"
```

With some editors, you may want to enforce asynchronous operation, because the editor has some problems when it runs the script directly. I had to do that with XDME:

```
GetEditPort:
IF Abbrev(Address(), "XDME.") = 1 THEN DO
    CALL rtezrequest "execute (run rx ASpell.xdme)", "_Abort", "Run this script  ↩
       asynchronously: ", rttags
    EXIT
END
IF ~SHOWLIST("P", "XDME.1") THEN DO
    CALL rtezrequest "XDME.1 unavailable", "_Abort", "Error:", rttags
    EXIT
END
RETURN "XDME.1"
```

1.94 GetScreen()

This function returns the name of the screen your editor is operating on. Most editors use the Workbench screen, and all you need for them is the following:

```
GetScreen: PROCEDURE
RETURN GETDEFAULTPUBSCREEN()
```

Some editors use their own custom screens. If your editor has some way of telling you the screen name, you can use that. For example, BlacksEditor lets you do this:

```
GetScreen: PROCEDURE
"GetScreenInfo"
```

```
info = result
screen = Word(info, Words(info))
RETURN Substr(screen, 2, Length(screen)-2)
```

Note that you do not need to enclose this with ADDRESS commands. This function is called while your text editor's port is the current port.

If your editor lacks this feature, you can just name the port:

```
GetScreen: PROCEDURE
RETURN "SkoEd"
```

1.95 Supported Text Editors

I have made ARexx scripts for several different text editors, but they are not all equal. Some text editors lack the features the GUI needs to work smoothly with the editor. Here are the text editors I have scripts for, grouped from BEST to WORST, with each group ordered alphabetically. Please note that these are appraisals of how well each editor works with the AlphaSpell GUI, not appraisals of the text editors themselves. I don't want to offend the authors of any of these products.

BEST: BlacksEditor, FrexxEd, GoldEd, GNU Emacs, Textra, TJM_DME, TurboText, and XDME

GOOD: Ed

ADEQUATE: Annotate

POOR: AmokEd, DME, and SkoEd

WORST: TKEd

Those that work best with the GUI perform whole word searching, run asynchronously, put the GUI on the same screen as the editor, operate smoothly, and allow the user to invoke spell-checking with menu item or internal command. Among these, BlacksEditor, FrexxEd, GoldEd, GNU Emacs, and TurboText use their own built-in routines for whole word searching. That makes searching faster for these editors, but there is occasionally a disparity between what AlphaSpell recognizes as a word and what the editor recognizes as a word. But the problem rarely arises, and the asynchronous operation of the GUI let's you handle it when it does. All the rest, as well as SkoEd, perform whole word searching with the help of ARexx. That makes it a bit slower, but that's compensated by the ARexx routine recognizing words in the same manner as AlphaSpell does.

Ed provides no facilities for an AlphaSpell menu item, because it has no command for invoking an ARexx script or running an external command. Annotate is only adequate for the GUI's purposes, because it doesn't have features sufficient for the implementation of whole word searching. AmokEd and DME share the same problem. Plus, they have difficulty indicating which word was just found when you search for a word. SkoEd lacks features for whole word searching, plus it pops up its own "Find & Replace Requester" whenever you want to search for a word. TKEd has the same problem as SkoEd.

Plus it won't save a temporary file. So, unless it can be rewritten, it can't actually be used with the AlphaSpell GUI.

I've included scripts for editors that don't cooperate well with the GUI in the hope that others might improve them, or that the authors might give their editors the features that they need to work well with the AlphaSpell GUI. I urge the authors of some of these editors, as well as the authors of other editors, to read my
note to text editor authors
.

1.96 A Note to Authors of Text Editors

If you would like the AlphaSpell GUI to work smoothly with your text editor, you should make sure that you includes ARexx commands functionally equivalent to the following pseudo code examples:

FIND WORD/K WHOLE/S CASE/S FIRST/S

This command searches for a word, and it includes options for whole word searching, for case sensitive searching, and for searching for the first instance of the word in the document.

Whole word searching can be left out if you include the ability for an ARexx script to read the current word. For example, XDME lacks a whole word search mode, but ASpell.xdme includes a routine for whole word searching. The advantage of doing it in the script is that it uses the same rules as AlphaSpell does to recognize a whole word.

REPLACE OLDWORD/K NEWWORD/K CASE/S

This command replaces the OLDWORD with the NEWWORD. The CASE switch tells it to give the NEWWORD the same case as the OLDWORD. For example, REPLACE cat Dog CASE would replace "cat" with "dog."

It is important for this command to work with the current word, because the GUI does interactive search and replace. First it finds the word, then it gives you the option of changing it. A command that just replaces the next instance of a word will not do.

DELCHAR N/A
INSERT TEXT/K

These commands will do in place of a REPLACE command. The first one deletes N characters. The second inserts the given text into the document. It is probably best that no word wrapping goes on when using such functions from the script.

SAVEFILE TO/A

This command saves the contents of the current buffer to the named file. If it changes the name of the file, you will also need the ability to find out the current file name, and the ability to change

```
    the file name. This will allow you to change the file name back to what
    it was after saving the buffer as a temporary file.

READ VAR/K

    This command stores the value of one of the editor's own variables
    into RESULT. Useful variables include a failure flag (which tells
    whether the last command failed), the file name, and the current word.

        To find out how functions such as these would work in a script,
read
                How to adapt the AlphaSpell GUI script for other text editors
                .
```

1.97 Why register AlphaSpell?

```
                 Ethical Reasons

                 Selfish Reasons

                 Another point of view
                SEE ALSO:
                How to Register AlphaSpell
```

1.98 Moral reasons for registering

```
                        I happen to be a professional ethicist. So I'm familiar ↩
                           with
different ethical theories. Here are reasons that some of them would give.
Take your pick.

        The Golden Rule

                Objectivism

                The Categorical Imperative

                Universal Prescriptivism
                SEE ALSO:
                How to Register AlphaSpell
```

1.99 The Golden Rule

```
        If you wrote some shareware, would you want people to pay you for
it? If you would, please do as you would have done to you and pay for the
shareware you use.
```

1.100 Objectivism

If you use this program regularly with no intention to pay for it, that is a violation of my property rights. To violate property rights is to live a secondhand existence and to ask people to live for your sake.

1.101 The Categorical Imperative

One formulation of the categorical imperative tells us to treat others as ends and never as means only. If you use shareware with no intention of paying for it, you are treating the author of the shareware as nothing but a means to your own well-being and productivity. But if you pay the authors of shareware you use, you are treating them not only as means but as ends.

1.102 Universal Prescriptivism

Consider the principle that it is morally permissible to use shareware without paying for it even when you can afford to pay for it. A few of you may follow this principle. But supposing you do, would you be willing to prescribe that everyone follow it? Do you think the world would be a better place if people regularly paid for shareware, or if people regularly used shareware without paying for it?

Certainly, someone may say, "There is just so much more good software available for me to use when I follow the principle of not paying for shareware." But would this still be true in a world in which no one paid for shareware? If no one paid for shareware, some people would still distribute freeware. But the incentive to write shareware would be gone. People who wanted to earn a living through programming would have to turn to commercial software instead. Commercial software would cost more, and it wouldn't be freely distributable. So if no one paid for shareware, less software would be freely distributable.

Of course, someone may reply, "So what? The world isn't like that. People do pay for shareware, and I'm not going to change that by not paying for shareware myself." And it is indeed true that other people will still pay for shareware even if some people refuse to. But those who do are taking advantage of a system whose existence requires that most people do not act like them. The shareware they use is available, because other honest people do pay for shareware. Such people leech off of a system that they don't maintain, and so act immorally.

1.103 Morality is for suckers

And now for a different view on shareware. For this opinion, I interviewed a big fat cat who named his dog after a tool of destruction, his daughter after a firearm, and his son after himself. I have withheld his name to avoid embarrassing the company he is associated with.

"Shareware authors are saps. They think people will pay them for stuff they can just take for free. I just use other people's shareware and laugh at how stupid they are to make it freely distributable."

But the registered version is better. So you might want the registered version anyway.

"I'll just pirate it when I can. Some goof is bound to actually pay for it."

But if everyone behaved like you, no one would register it.

"What kind of fool do you take me for? I don't want other people to behave like me, you dipstick! I like being smarter than everyone else. That's how I get ahead!"

So you're saying it's stupid to be moral?

"That's right. Morality is for suckers. Anyone with half a brain is out for #1."

1.104 What you get for registering

The Keyfile

Intangibles

1.105 What I send you when you register

When you register AlphaSpell, I will email you your keyfile in uuencoded form. There are plenty of programs available for the Amiga that will decode it for you. Alternately, for an extra $5.00, I will mail you a disk with your keyfile on it. The keyfile is a mostly binary file that identifies who you are and gives you access to all of AlphaSpell's features. With the keyfile, you will be able to write compressed dictionaries, and you will be able to set any value for the edit distance with the MATCH command.

Your keyfile will let you use the registered features in future updates to AlphaSpell. For security reasons, I reserve the right to change the keyfiles when needed. But when I do, registered users will receive new keyfiles free of charge. I will change the keyfiles if I learn that someone's keyfile has gotten into someone else's hands. If that happens, the user with the errant keyfile will no longer be registered.

1.106 What else you get for registering

When you register AlphaSpell, you not only get the keyfile; you also encourage me to put more work into AlphaSpell, making sure that it is a quality product that meets your needs. When I write Freeware, I write it mainly for myself, I don't test it thoroughly, and I don't put as much effort into making sure it is a quality product. Take XES, for example. It makes XDME a lot easier for me to use, especially since I wrote it and know what everything does, but it isn't documented fully, and you will have to study it at the source code level to get the most out of it. Or take the DDLI. I haven't updated that in a long time. People seem to like it, but I could devote more time to it if I wished and make it even better. I just work on it when the mood strikes me, and although I've modified it beyond the latest release, I haven't gotten around to releasing the new version.

But things are altogether different with AlphaSpell. This is shareware. So I try to make it as good a product as I can, because I want you to use it and pay me for it. I'm interested in propagating its use around the globe. So I've made around a dozen AlphaSpell dictionaries for different languages, most of which I'll never need myself. When you pay for AlphaSpell, you will be telling me that my efforts are well directed, that I should continue making AlphaSpell better and better, and that I should continue to make sure that AlphaSpell is the best spelling checker you can use.

1.107 How to Register AlphaSpell

AlphaSpell itself is SHAREWARE. You can register AlphaSpell for $20.00 in American currency. You can register with cash, check, or credit card. To register with cash or check, send $20.00 in American currency, or a $20.00 check drawn from an American bank, to:

Fergus Duniho
1095 Genesee St.
Rochester, NY 14611-4148

To register with your Master Card, Visa, Discover, or American Express card, you have to register on-line through BitNova. You can register by telneting to BitNova directly or through BitNova's World Wide Web Site. BitNova is accessible through the AlphaSpell web page:

http://www.bitnova.com/duniho/

1.108 About the Author

I am a 29 year old graduate student in the Philosophy Department at the University of Rochester. This past semester, I have been am teaching a course on the nature of evil, and next Summer I will be teaching a course

on computer ethics. I am also the author of a personality indicator known as the DDLI.

If you're on the Web, check out my homepages:

http://www.ling.rochester.edu/~duniho/index.html
http://www.geocities.com/Athens/4723/

The first is my school home page, which will disappear after I graduate, but the second is unaffected by my location. It should still be around when the other is gone.

Also check out the AlphaSpell support page:

http://www.bitnova.com/duniho/

It contains links to AlphaSpell, its GUI's, and its dictionaries, as well as to stuff used by the AlphaSpell GUI. Anything I release of my own will show up here before it shows up on the Aminet.

Registered
users

will be emailed about updates as soon as they're available.

If you have comments, questions, or suggestions, email them to:

duniho@bitnova.com

Although I have other email addresses I use more often, this address will outlive them. It is the address associated with AlphaSpell's official support site. Any mail you send there will automatically be forwarded to one of the email accounts I have at school. When I finally graduate and get a real job, which may be in a year or so, I will have new email addresses. My current addresses are:

fdnh@troi.cc.rochester.edu
fdnh@ro.cc.rochester.edu
fdnh@picard.cc.rochester.edu
fdnh@riker.cc.rochester.edu
fd001d@uhura.cc.rochester.edu
duniho@www.ling.rochester.edu

The top four addresses are all really the same. Uhura and BitNova currently forward stuff to duniho@www.ling.rochester.edu.

My mailing address is

Fergus Duniho
1095 Genesee St.
Rochester, NY 14611-4148
USA

I expect it will be good for at least another year, and I expect my current email addresses will be good for the same amount of time. If you want to make sure that you have my current address, look me up on the World Wide Web.

1.109 History

1.110 The Ancient History of AlphaSpell

AlphaSpell has its roots in a spelling checker I wrote in ARexx back in Fall 1991 or Spring 1992. I learned C in the Spring of 1992 and rewrote that spelling checker in C for my first major project in C. Initially, it was mainly a brute force spelling checker. Its main virtue was that it finished spell checking quickly, because it spell checked words in alphabetical order, thereby requiring only one pass through the dictionary.

AlphaSpell 2 was also in C, but I never released it. It had the ability to use a compacted dictionary.

AlphaSpell 3 was a new C++ program. It implemented the same basic algorithm as V2.00, but with less redundancy. This version also did the tasks that previous versions depended on other programs to do. Previous versions required other programs to get the words from a file, to sort those words, and to remove redundancies, in order to create an alphabetized list that AlphaSpell could read. AlphaSpell V3.00 did all this on its own.

AlphaSpell 4 is in C again, because C++ has changed, and some of my old C++ code is broken. Unfortunately, I don't have references on the latest revision of the language. So I translated it all back into C and added a bunch of new features. New features include word counting, guessing, testing, weeding, and the ability to work with multiple dictionaries, both compressed and uncompressed. Previous versions expected input from standard input. Version 4.00 does not unless you tell it to by using "stdin" as a file name.

AlphaSpell 5 is a C++ program again, and the code is just about a complete reworking of AlphaSpell 4.00's code. New features include a UNIX like interface, new guessing algorithms, a new compression format for dictionaries, pattern matching, and a new GUI for XDME. The new guessing algorithms are based on sounder methods than I employed in the last version. One measures phonetic similarity between words, and the other checks how easily one word can be changed into another.

AlphaSpell 6 is a C program again, because my C++ code wasn't working well with GCC 2.7.0. This version is another facelift. It replaces the UNIX interface of AlphaSpell 5 with an AmigaDOS ReadArgs interface. Each of AlphaSpell's commands has a different ReadArgs template. AlphaSpell now sports an ARexx port, which includes AlphaSpell's standard commands plus some others useful mainly in ARexx scripts. Many Standard C library

functions have been replaced with AmigaDOS functions. SH pattern matching
has been replaced with AmigaDOS pattern matching. Whereas previous versions
took no advantage of AmigaDOS, AlphaSpell 6 takes advantage of many
AmigaDOS features.

1.111 Revisions of AlphaSpell since 6.0

```
ABBREVIATIONS: BF = Bug Fix, NF = New Feature, CF = Changed Feature
               CM = Code Modification, OP = Optimization

6.1 (2 May 1996)

    AlphaSpell's Web page was incorrectly named in 6.0. Fixed that.

6.2 (10 May 1996)

    TeX support added. The TEX switch makes certain commands discard TeX
    commands.

    The HOMOPHONES/S option of the MATCH command was removed. Its presence
    caused AlphaSpell to require translator.library even if the option
    wasn't used, and its neglibible utility didn't justify fixing
    AlphaSpell so that it wouldn't be a problem. The HOMOPHONES option was
    damn slow and didn't give you much in return.
```

1.112 History for the AlphaSpell GUI

```
ABBREVIATIONS: BF = Bug Fix, NF = New Feature, CF = Changed Feature
               CM = Code Modification, OP = Optimization

2.0

    NF - Communicates with AlphaSpell through its ARexx port
    NF - Select language with listview in Prefs window.
    CF - New design for the Main Window
    NF - Names registered users at startup.
    NF - Makes use of some of AlphaSpell's new ARexx commands.

2.1 (2 May 1996)

    BF - Changing the language no longer causes problems.

2.2 (10 May 1996)

    BF - The script didn't correctly get the text editor's port name.
         Fixed.

    NF - Languages are now alphabetized in the Preferences listview.

    CM - Scripts that used "Clipsave" to save the clipboard to a file now
         use some functions from RexxTricks.library.
```

```
    CF - ASpell.textra uses RexxTricks.library functions to save the
         clipboard to a file. It previously copied the file to a new
         window, saved the contents of the new window, and closed it. This
         was a kludge in my opinion.

    NF - The script exits if the version of RexxTricks.library is too low.

    NF - Preferences are now stored as tooltypes in the ASpell.gui icon.
         The ASpell.prefs file is obsolete.

    NF - If the argument "CLIP" is passed to the script, it will spell
         check the contents of the clipboard instead of the current
         document.

    NF - If the argument "TEX" is passed to the script, it will ignore TeX
         commands during spell checking.
```

1.113 Credits and Acknowledgments

AlphaSpell is a program by Fergus Duniho. I compiled it with GCC and libnix. Thanks go to Andy Cook for arexxport.library, which AlphaSpell uses for its ARexx port.

Thanks go to Michal Kara for DED (a.k.a. Disk-Editor) and to Rainer Koppler for Cvt. DED helped me recover one of the dictionaries and most of this file when I accidently overwrote them. Cvt helped me convert some wordlists I found to ISO Latin.

1.114 Dictionaries available for AlphaSpell

To date, AlphaSpell dictionaries are available for the following languages:

Afrikaans

Danish

Dutch

English

French

German

Icelandic

Latin

Norwegian

Spanish

Swedish

Previously, this documentation said that an Italian dictionary would be available. Unfortunately, the Italian word list I have lacks accents. So I've chosen not to use it.

1.115 Afrikaans Dictionaries

FILE: Afrikaans.lha
SHORT: Afrikaans dictionary for AlphaSpell
Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Fergus Duniho and Bernard Nieuwoudt
Version: 1.1

This is an Afrikaans AlphaSpell dictionary based on:

ftp://sable.ox.ac.uk/pub/wordlists/afrikaans/afr_dbf.zip

Bernard Nieuwoudt, the author of the original file, says "If the user of the list makes alterations to the list, the list then becomes the property of that user. I then relinquish all rights (and all responsibility) of any sort to the list." Since I have made alterations to the original list, the ownership of the new list falls to me, Fergus Duniho. The original list contained words with spaces, such as "a priori" and "a capella-koor," but the new list does not contain spaces in any words. It contains "priori" and "capella-koor," which the original list doesn't. I made this alteration, because AlphaSpell recognizes the space as a word delimiter. It can't tell you whether "a priori" is spelled right, but it can tell you whether "priori" is.

What follows is the text from Bernard Nieuwoudt's original readme file:

This is a message for the accompanying file AFR_DBF.ZIP:

That file contains a list of Afrikaans words. It is PKZIP-ed (PKZIP 2.04g) and posted in binary from a DOS platform.

This list was compiled in personal capacity since 1984. Many works were referenced, but the major part of the list was compiled by myself.

Users of the list agree, by using it, to the following:

1) The list is used entirely at own risk. I will not be held liable for any mistakes, ommisions, law suites etc.
2) The use of the list as it was posted, is for personal use only.
3) As there were many sources for the list, credit cannot be given to all sources. It is suggested that any commercial use of part of the list should first be vetted by lawers.

4) Commercial use of the list, as it is, should first be cleared out with me.
5) If the user of the list makes alterations to the list, the list then becomes
   the property of that user. I then relinquish all rights (and all
   responsibility) of any sort to the list.

I hope this makes sence and that you find it agreeable?

Greetings
Bernard Nieuwoudt

TEL: (012) 420 3637
EMAIL: BERNARD@CCNET.UP.AC.ZA

1.116 Danish Dictionaries

FILE: Danish.lha
SHORT: Danish dictionary for AlphaSpell
Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Fergus Duniho and Unknown
Version: 1.1

This is a Danish AlphaSpell dictionary based on:

ftp://sable.ox.ac.uk/pub/wordlists/danish/danish.words.Z

I don't know who originally compiled the wordlist. The original wordlist was entirely in 7-bit ASCII, using what I took to be the Denmark II character set. So I wrote a conversion script for Rainer Koppler's program Cvt, which translates ASCII characters in the Denmark II character set to their 8-bit equivalents in ISO Latin. That script is included in this package.

Please note that the wordlist was entirely in lowercase, and a cursory examination of the file suggests that names have been included in lowercase. For example, I saw the string "anna" in the file. So this package does not include a mixed case dictionary for case sensitive checking.

1.117 Dutch Dictionaries

FILE: Dutch1.lha
SHORT: Dutch dictionary for AlphaSpell
Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Fergus Duniho and Erik Frambach
Version: 1.1

This dictionary is based on an archive I found on SimTel called nlword10.zip. The original archive contains 26 files, nl.a, nl.b, ..., nl.z. These were plain text files using the IBM character set, and sorted alphabetically. To construct this dictionary, I removed the carriage

returns, converted the non-ascii characters from IBM to ISO, resorted the files, and fed them all into a single file. The total number of words in the dictionary is approximately 220,000.

If you find any mistakes in the dictionary, don't write to me. I don't know any Dutch. This dictionary is not maintained by me. It based on a dictionary maintained by Erik Frambach. Please send any comments on misspelled or missing words to him. His address is:

E.H.M.Frambach@eco.rug.nl

Here is what he wrote in his original readme file:

This Dutch dictionary contains 26 files, NL.A, NL.B, ..., NL.Z.
Each file contains words that start with the letter indicated by
the file extension.
The files are plain text files, the words are in extended ASCII,
each word is on a separate line, and they are sorted alfabetically
per file. The total number of words is approximately 220,000.

This is version 1.0 of the Dutch dictionary.
Please send any comments on misspelled or missing words to
E.H.M.Frambach@eco.rug.nl

1.118 English Dictionaries

FILE: ukacd.lha
SHORT: Big English dictionary for AlphaSpell
Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Fergus Duniho and Ross Beresford

This dictionary is based on version 1.3 of Ross Beresford's "UK Advanced Cryptics Dictionary." This is "a word list for crossword solvers and setters." The main differences between Ross Beresford's original dictionary and the AlphaSpell dictionary are these. (1) The AlphaSpell dictionary is one file, and the original dictionary is 26 files. (2) The AlphaSpell dictionary is compressed, and the original dictionary is straight ASCII. (3) The original dictionary contained phrases as well as words, but the AlphaSpell dictionary does not contain the phrases. This is because AlphaSpell is primarily a spelling checker, and it only checks words against words, not against phrases.

Although AlphaSpell is primarily a spelling checker, it can also be used to solve crossword puzzles. For example, if you're looking for a five letter word whose second letter is p, you could send the pattern "?p???" to AlphaSpell. AlphaSpell is also useful for solving scrambled word puzzles.

1.119 French Dictionaries

FILE: French.lha
SHORT: French dictionary for AlphaSpell

Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Fergus Duniho and Unknown
Version: 1.1

This AlphaSpell dictionary is based on the French dictionary that comes with the Unix version of a program called "Le Dico," which is available as:

file://cipcinsa.insa-lyon.fr/apps/pub/france/ledico_u.zip

This version was already in Latin ISO. Based on my poor knowledge of French, Le Dico seems to do some dictionary maintenance and pattern matching. I don't know whether it does anything else.

Le Dico came with a file called "copying.doc". I asked someone who knows French to tell me what it said, and then I looked up words in a French-English dictionary to bolster my understanding of it. As I understand it, I am free to use the dictionary as I wish. If I were modifying the source code to Le Dico or to its utility programs, I would have to include the complete sources. But I am not doing that. AlphaSpell may do some of the same things as Le Dico, but it is not based on it in any way. All I'm doing is making Le Dico's lexicon available for AlphaSpell to use.

For those who know French, (and why would you want this dictionary if you don't?), here is the French text on distributing Le Dico:

```
   +--------------------[ Distribution De ]-----------------------+
   |                                                              |
   |    LL       EEEEEEE        DDDDDD   IIII   CCCCC    OOOOO    |
   |    LL       EE             DD   DD   II   CC   CC  OO   OO   |
   |    LL       EEEE           DD   DD   II   CC       OO   OO   |
   |    LL       EE             DD   DD   II   CC       OO   OO   |
   |    LL       EE             DD   DD   II   CC   CC  OO   OO   |
   |    LLLLLLL  EEEEEEE        DDDDDD   IIII   CCCCC    OOOOO    |
   |                                                              |
   +--------------------------------------------------------------+
```

Le Dico n'est pas domaine public, il est "Freeware".

Considerez que Le Dico et les fichiers le composant sont distribues pratiquement avec un "CopyLeft" similaire à celui des programmes GNU, parceque c'est un exemple du genre. (Mais Le Dico n'a rien a voir avec GNU bien entendu).

En resume, vous etes libres de diffuser gratuitement Le Dico a qui vous voulez. Vous etes libres d'utiliser tout ou partie des sources et fichiers le composant pour toute realisation, le lexique est d'ailleurs fait pour cela !

Distribution:

Vous devez fournir l'archive originale complete, sans jamais dissocier les fichiers la composant. La version d' origine est la version Unix, faites circuler cette version universelle de preference a l'adaptation DOS chaque fois que c'est possible.

Si vous faites des modifications ou ameliorations, ou realisez des utilitaires a distribuer avec Le Dico, vous devez le signaler dans la documentation, et fournir les SOURCES COMPLETES (et portables dans la mesure du possible) de tout ce que vous implementez, ou du moins laisser ces sources disponibles a tous gratuitement, sur simple demande.

Ceci est indispensable pour le developpement de ce type de programme 'ouvert' et gratuit, devant etre accessible a toute la communaute.

1.120 German Dictionaries

FILE: German.lha
SHORT: German dictionary for AlphaSpell
Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Fergus Duniho and Unknown
Version: 1.1

Dies ist ein deutsches AlphaSpell Wörterbuch.

This is a German AlphaSpell lexicon based on:

ftp://sable.ox.ac.uk/pub/wordlists/german/germanl.Z

I don't know who originally compiled the wordlist. The original wordlist was entirely in 7-bit ASCII, using `"' after vowels to indicate umlauts, and `sS' to indicate `ß'. So I wrote a conversion script for Rainer Koppler's program Cvt, which translated it to ISO Latin. That script is included in this package.

I came across a larger German dictionary at the same site, which had German words written entirely in Roman characters. Umlauted vowels were followed by an e, and ss replaced ß. I've translated the umlauts, but I don't yet know how to decide which s-pairs should be converted into ß. Once I've learned how, I'll finish translating it and upload it.

1.121 Italian Dictionaries

If you really need an Italian dictionary, you can download and adapt:

ftp://sable.ox.ac.uk/pub/wordlists/italian/words.italian.Z

But you should be warned that this wordlist is missing accents. It is for this reason that I have chosen not to base an AlphaSpell dictionary on it.

1.122 Latin Dictionaries

```
FILE: Latin.lha
SHORT: Latin dictionary for AlphaSpell
Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Fergus Duniho and Unknown
Version: 1.1
```

This is a Latin AlphaSpell dictionary based on:

ftp://sable.ox.ac.uk/pub/wordlists/latin/wordlist.aug.Z

1.123 Norwegian Dictionaries

```
FILE: Norwegian.lha
SHORT: Norwegian dictionary for AlphaSpell
Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Fergus Duniho and Unknown
Version: 1.1
```

This is a Norwegian AlphaSpell dictionary based on:

ftp://sable.ox.ac.uk/pub/wordlists/norwegian/words.norwegian.Z

The README file in the wordlists directory seems to indicate that this wordlist was compiled by Anders Ellefsrud <anders@ifi.uio.no>. The original wordlist was entirely in 7-bit ASCII, using what I took to be the Norway character set. So I wrote a conversion script for Rainer Koppler's program Cvt, which translates ASCII characters in the Norway character set to their 8-bit equivalents in ISO Latin. That script is included in this package.

1.124 Spanish Dictionaries

```
FILE: Espanol.lha
SHORT: Spanish dictionary for AlphaSpell
Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Joan Sola
Version: 1.1
```

This dictionary was provided by Joan Sola, who in providing it became the first registered user of AlphaSpell. On this dictionary, Joan Sola writes, "I work as a translator (English -> Spanish) and this dictionary is the result of some massive extraction of words from texts. Spelling is good, since I spell checked this dic text in a PC wordprocessor."

1.125 Swedish Dictionaries

FILE: Swedish.lha
SHORT: Swedish dictionary for AlphaSpell
Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Fergus Duniho and Unknown
Version: 1.1

This is a Swedish AlphaSpell dictionary based on:

ftp://sable.ox.ac.uk/pub/wordlists/swedish/words.swedish.Z

I don't know who originally compiled the wordlist. The original wordlist was entirely in 7-bit ASCII, using what I took to be the Swedish character set. So I wrote a conversion script for Rainer Koppler's program Cvt, which translates ASCII characters in the Swedish character set to their 8-bit equivalents in ISO Latin. That script is included in this package.

1.126 Icelandic Dictionaries

FILE: Icelandic.lha
SHORT: Icelandic dictionary for AlphaSpell
Type: text/edit
Uploader: fdnh@troi.cc.rochester.edu
Author: Jorgen Pind
Version: 1.1

This dictionary was made from an Icelandic word list provided to me by Arni Freyr Jonsson in return for AlphaSpell's keyfile. The wordlist was made by Jorgen Pind for a program called stafs, which he made while working at "Orðabók Háskólans," The University of Iceland Dictionary. Jorgen Pind writes in Icelandic:

Það er guðvelkomið, orðalistinn er í public domain eða því sem næst. Sjálfur er ég höfundar listans og útbjó hann þegar ég starfaði á Orðabók Háskólans. Geta mætti þessa ef þið látið upplýsingar fylgja orðaskránni.

Kveðja,

Jörgen

Jorgen Pind Tel. +354-525-4086 Fax. +354-552-6806
Department of Psychology
University of Iceland Internet: jorgen@rhi.hi.is
Oddi, 101 Reykjavik, Iceland

As translated into English by Arni Freyr Jonsson, this reads:

Yes, please do use my wordlist. It is practically public domain. I am the author of the wordlist myself and made it while working at The University of Iceland Dictionary. Please include this information with the

distribution of your program.

1.127 Making a dictionary

There are basically two steps to creating an AlphaSpell dictionary:

(1)
Acquiring a wordlist
(2)
Converting a wordlist

1.128 Acquiring a wordlist

There are basically three ways to acquire a wordlist:

Write one from scratch

Generate one from word frequencies

Find one by someone else

1.129 Writing a wordlist from scratch

The most effective way to do this is to type in words from a paper bound dictionary. This is a good way to create an accurate dictionary, but you had better be a fast typist to make the effort worthwhile.

1.130 Generating a wordlist from word frequencies

This is something AlphaSpell is designed for. It allows you to tabulate the word frequencies from countless reams of literature for creating a wordlist of commonly used words. There are many places on the Internet to find works of literature free for downloading. If you take this approach to creating a dictionary, here are some things to bear in mind. All the files you cull word frequencies from should be in the same language, and they should all use the same character set, preferably ISO Latin-1. If special characters are expressed with HTML codes or some other sort of coding, you should convert the files to a genuine character set before proceeding further. HTML codes and the like would screw things up, giving you misspelled words.

Suppose you download a bunch of literature from the Internet and put it all in a directory called lit. You could then type:

```
AlphaSpell TALLY lit/* TO freqlist
```

This would create a list of the word frequencies in the files. If you downloaded as many files as you had disk space for and want to add more, you can now delete the files, download more, and create a second list of frequencies. Using the LIST switch, you can merge the lists into one master list, then delete stuff and download more, repeating this cycle until you have a large enough list of word frequencies. You can merge frequency lists together like so:

```
AlphaSpell TALLY freqlist flist2 TO freqlist LISTS
```

If you use the LISTS switch, you should randomize the lines in the files first, so that AlphaSpell will read them much faster.

Once you have a large frequency list, you can use the WEED command to create a list of common words like so:

```
AlphaSpell WEED freqlist TO common FREQ 4000
```

1.131 Finding an already available wordlist

There are many wordlists available in:

ftp://sable.ox.ac.uk/pub/wordlists

If you're a college student there may be a wordlist available on the machine you read email on. But before you try to distribute it as an AlphaSpell dictionary, you should make sure there are no copyright restrictions on it that would prevent you from doing so.

You may find various dictionaries and wordlists on the Internet by entering "spell" or "dictionary" into a search engine such as SHASE or archie.

1.132 Converting a wordlist

Once you have a wordlist, you need to convert it into an AlphaSpell dictionary. Here is a guided example on how to do this. Suppose you come across a dictionary for Spangalese while exploring the web sites in Spanga.

The first thing you should do is make sure that it is in a character set recognized by AlphaSpell. AlphaSpell is designed to use and recognize the ISO Latin-1 character set. So make sure the wordlist uses this character set. If it doesn't convert it.

As it turns out, the Spangalese wordlist you found uses the IBM character set. To convert it, you may use cvt like so:

```
cvt Spangalese DSC DOSToAmi.cvt
```

Since the original DOSToAmi.cvt was missing some conversions, I've included an improved version with AlphaSpell.

Next, do a case sensitive sort on it. I use FSort and would type:

fsort Spangalese Spangalese case

Next, put all words with uppercase letters into Spangalese.amd by typing:

AlphaSpell MATCH *[A-ZÀ-Þ]* Spangalese TO Spangalese.amd CASE

The wildcard pattern includes the set of all capitalized letters. In ISO Latin-1, these are all the letters from A to Z and all the letters from À (capital A with a grave accent) to Þ (capital thorn). On the Amiga, you enter À by typing alt-g followed by A, and you enter Þ by typing alt-T.

Put the rest of the words into Spangalese.ald by typing:

AlphaSpell MERGE Spangalese Spangalese.amd SUB TO Spangalese.ald

After these steps, you should have a lowercase Spangalese dictionary, Spangalese.ald, and a mixed case Spangalese dictionary, Spangalese.amd.

1.133 AlphaSpell Support

AlphaSpell has an official support site on BitNova.

BitNova
Telnet: bitnova.com
Phone : (510)581-0600
Web : www.bitnova.com
FTP : ftp.bitnova.com

You can also visit the AlphaSpell homepage:

http://www.bitnova.com/duniho/

This page includes links to the latest versions of AlphaSpell, this GUI, and the dictionaries. The newest stuff is always available here before it is available on the Aminet.

1.134 Installing AlphaSpell

The accompanying installation script takes care of the greater part of the installation. However, there is one detail you should attend to manually. To make it easier to use AlphaSpell, it helps to have some menu items that invoke AlphaSpell automatically. This usually involves editing

the configuration file of your text editor. Here is information on including such a menu item for individual text editors.

NOTE: The following examples assume that the script is in REXX:. If it is in AlphaSpell:REXX/ or someplace else, you should include the full path name.

NOTE: Some of the editors require that you run the script asynchronously by executing the script with the rx shell command, which is done in turn by running rx with run. If you have runback available, use that instead of run.

NOTE: The examples are for a menu item that spell checks the current document. Depending on your needs you may also want to add menu items for spell checking the clipboard, TeX files, and the clipboard as a TeX file. To add these other menu items, repeat the steps described for your text editor and include the appropriate argument after the script's name. Use "CLIP" for checking the clipboard and "TEX" for checking TeX files. Use both for checking the clipboard as a TeX file.

AmokEd, DME, TJM DME, XDME

These are all related to each other and will use the same code for the menu item. AmokEd's configuration file is called .aedrc, and DME, TJM DME, and XDME all use a configuration file called .edrc. You should insert the following line in the configuration file:

```
menuadd AlphaSpell (Check Spelling) (execute (run rx ASpell.xdme))
```

Annotate

Add the following two lines to S:AnnTools:

```
Spell Check
run rx ASpell.ann
```

It is important to run Annotate from the CLI. Otherwise, Annotate won't know where to find the programs in SYS:Rexxc/.

BlacksEditor

Edit the Support/Startup.dfn file. Goto the ARexx menu in the menus section and insert the following line someplace in there:

```
ITEM  "Spell Check"          ""  ExecARexxMacro ASpell
```

Ed

I don't think it can be done for Ed.

Emacs

To put the AlphaSpell menu item in Emacs, you need to edit the file .emacs-menu.el. This file contains a single amiga-menus-set command. To add a menu item, you should insert the following code just before the last parenthesis of the amiga-menus-set command:

```
(quote
    (("AlphaSpell"
        (("Spell Check"
            (shell-command "run rx ASpell.elx")
            nil))
    ))
)
```

FrexxEd

    To add a menu item to FrexxEd, add the following code to User.FPL:

```
    MenuAdd ("t", "AlphaSpell");
        MenuAdd ("i", "Spell Check", "System(\"run rx ASpell.rx\");");
```

GoldEd

    For GoldEd, you don't need to edit a configuration file. Instead, select the "Menus..." item in the "Config" menu. This will present you with a requester for editing menu items. At the left, add a menu called "AlphaSpell". In the middle, add an item called "Spell Check". Then double click where it says "Spell Check" in the listview. This will open up the "event definition" requester. Select the ARexx button at the right. In the listview, add the name of the script and put quotation marks around it. The script's name is ASpell.ged. Click on the "OK" gadget of the "events definition" requester. Click on the "Save" button of the "Menus" requester and save to the appropriate configuration file. Click on the "OK" button of the "Menus" requester and you're done.

SkoEd, TKEd

    These editors work poorly with AlphaSpell. So I didn't bother to find out whether menu items could be made for them. I'm not sure that they can.

Textra

    Textra gives you an ARexx requester in which you can put the names of different ARexx scripts. Type in "ASpell" for one of them and save it.

TurboText

    For TurboText, add the following code to the appropriate menus definition file, such as TXX_Menus_English.dfn for English speakers.

```
  MENU AlphaSpell
    ITEM  "Spell Check"          ""  ExecARexxMacro ASpell
```

1.135 Index

Universal Prescriptivism

Usage of AlphaSpell

Using the AlphaSpell GUI
W

What else you get for registering

What I send you when you register

What you get for registering

Why register AlphaSpell?

Writing a wordlist from scratch