How To:

Understanding Classic SoundEx Algorithms

Search Names & Phrases Based on Phonetic Similarity (w/source code)

Use our SoundEx Generator Form below.

[top]

Overview & History

Documents
Tech/How-to
- Understanding SoundEx
- CSV File Format
- Temporality
- Dell Service Tags
- Round Table Corners
- Menu Flowers (exercise)
- Show XML w/I.E. Browsers

Conventions
- **CTX™ File Format**
- Hybrid Oriented Programming
- Software Stability Ratings
- The JSMsg Convention
- Glosssary XML
- CUF™ Codes

Managing IT
- The Un-Methodology
- Continuous Independence

º Agile Bridge Building
º Platform Neutral Computing

Random Notes
º Seeing Through Hype

Terms that are often misspelled can be a problem for database designers. Names, for example, are variable length, can have strange spellings, and they are not unique. American names have a diversity of ethnic origins, which give us names pronounced the same way but spelled differently and vice versa.

Words can be misspelled or have multiple spellings, especially across different cultures or national sources.

To help solve this problem, we need phonetic algorithms which can find similar sounding terms and names. Just such a family of algorithms exist and have come to be called SoundEx algorithms.

A Soundex search algorithm takes a written word, such as a person's name, as input and produces a character string that identifies a set of words that are (roughly) phonetically alike. It is very handy for searching large databases when the user has incomplete data.

The original Soundex algorithm began in a patent by Robert C. Russell in 1918. The name "Soundex" came along later (it was probably coined by the telephone company but I'm not sure). The original patent was simply titled "*Index*" (application Number 1,261,167, filed Oct. 25, 1917). Other patents by Russell involved index systems and gadgets, and some were variations and improvements of the 1917 index system:

- 1,261,167 (*Index*, filed 1917)**
- 1,357,653 (*CARD-INDEX*, filed 1919)
- 1,478,765 (*CARD-INDEX*, filed 1922)
- 1,478,672 (*CARD-INDEX*, issued 1923)
- 1,548,752 (*CARD-INDEX*, filed 1923)
- 1,548,753 (*CARD-INDEX*, filed 1923)
- 1,601,925 (*CARD-INDEX*, filed 1924)

There are considerable and credible sources on the Web (as well as the usual unreliable sources) naming one Margaret O'Dell as a co-inventor, but I have not been able to find any actual evidence of this. If you have information showing Margaret O'Dell to be a co-inventor of any of these patents, please **contact me**. so that she can be given credit here for her contributions.

Also the patent filed on 1917 (1,261,167) is not a perfect match but is considered the first Soundex because it is very similar to the algorithms we use today. Russell did produce at least one earlier index system that assigned numbers to "key-letters" which he had filed a patent application for a couple of years earlier. Finally, Russell's patent number 1,261,167 was reissued as # Re.15,582. None of these patents, as far as I can tell, list any inventor named O'Dell.

The method used by Soundex is based on the six phonetic classifications of

human speech sounds (bilabial, labiodental, dental, alveolar, velar, and glottal), which in turn are based on where you put your lips and tongue to make the sounds.

The algorithm itself is fairly straight forward to code and requires no backtracking or multiple passes over the input word. In fact, it is so straight forward, I will start by presenting it as an outline. I will continue on to give C, JavaScript, Perl, and VB code as well later.

[top]

The Algorithm as an Outline

1. Capitalize all letters in the word and drop all punctuation marks. Pad the word with rightmost blanks as needed during each procedure step.
2. Retain the first letter of the word.
3. Change all occurrence of the following letters to '0' (zero):
   'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
4. Change letters from the following sets into the digit given:
   - 1 = 'B', 'F', 'P', 'V'
   - 2 = 'C', 'G', 'J', 'K', 'Q', 'S', 'X', 'Z'
   - 3 = 'D','T'
   - 4 = 'L'
   - 5 = 'M','N'
   - 6 = 'R'
5. Remove all pairs of digits which occur beside each other from the string that resulted after step (4).
6. Remove all zeros from the string that results from step 5.0 (placed there in step 3)
7. Pad the string that resulted from step (6) with trailing zeros and return only the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

[top]

Discussion

The algorithm presented above is slightly improved over the originally patented algorithm.

The original Soundex algorithm of 1918 starts to fail when the number of words in the database gets to be large. For example, the diversity of names in a large database with many foreign spellings starts to put more and more phonetically unlike names into the same code.

The *slightly improved* algorithm presented here differs from the original in step three, where the vowel sounds are replaced with zeros ('0'), and in step (6.0) where those zeros are removed. The original algorithm eliminated the vowels completely in step 3.0, before duplicate adjacent digits are removed.

This improved version will produce SoundEx codes with duplicate digits in them; for example, "HERMAN" will code to "H06505" which will then reduce to "H655" in the last step. In the original version, "HERMAN" would code to "H655" which will then reduce to "H65" and finally to "H650" in the last step.

Other SoundEx algorithms exist as well, such as LEAA codes used in crime prevention databases, and Cutter Tables used by libraries to encode author names. Each has its advantages and disadvantages. For example, Cutter Tables have as an advantage, that the codes produced maintain the alphabetical order of the original material.

[top]

Enhancements

It's not hard to think of improvements that will make this already powerful algorithm even more robust. An example (at least to American pronunciation sensibilities) might include replacing many multi-letter sequences that produce unrelated sounds before performing the steps of the basic algorithm. For example, before starting the above procedure, replace:

- **DG** with **G**
- **GH** with **H**
- **GN** with **N** (not 'ng')
- **KN** with **N**
- **PH** with **F**
- **MP** with **M** ...WHEN... it is followed by S, Z, or T
- **PS** with **S** ...WHEN... it starts a word
- **PF** with **F** ...WHEN... it starts a word
- **MB** with **M**
- **TCH** with **CH**

- **A or I** with **E** WHEN - starts word+followed by[AEIO]
  (Not in source code...)

The conversion enhancement for **PF** would not normally be needed because both letters are in the same group (group 1). However, since this conversion improvement is only for the start of the word, it must be included, since the first letter is preserved in this and classic SoundEx.

Challenge: Can you think of more ways to improve this basic SoundEx algorithm?

[top]

SoundEx and the Census

The U.S. census has been making use of SoundEx codes to index surnames since the late 1800's. Those doing census lookups must use the same method to encode surnames as the census takers did when they generated the database. That means, for starters, our clever set of enhancements can't be used.

With one exception, the 'normal' method of encoding the census soundex codes is identical to that described in the **Algorithm as Outline** section above. The exception is a special rule for the letters **H** and **W**.

In the method described above, if two letters from the same group are on either side of a vowel, an 'H', or a 'W', they are considered two separate letters and they are not removed by the 'remove adjacent digits' step (5).

But in 'normal' census SoundEx, the 'H' and 'W' were completely removed from the conversion when surrounding letters were in the same group, so letters from the same group were combined into a single digit when only 'H' or 'W' were between them. For example:

***normal* census method:**

- **Ashcroft = A261**
  The 'S' and 'C' are both from group 2, and the 'H' is not considered a vowel-like separator, so the two are considered adjacent 2s which are combined into a single 2.
- **Asicroft = A226**
  Here, since vowels are considered separators, the two letters from group 2 are not adjacent and so do not get turned into a single 2.

Ok, that is the ***normal*** method that was used by the census to produce SoundEx codes. Now comes a little twist. Prior to the 1920 census these

codes were calculated by hand and many census takers ignored the special rule for 'H', and 'W'. That means they wrote their census codes exactly as described in the Algorithm as Outline section above. In other words:

***special* census method:**

- **Ashcroft = A226**
  No special rule was applied for the psuedo-vowels H or W so 'shc' was interpretted as two separated (not adjacent) letters, hence there are two '2' digits in a row.

So the census for 1880 through 1910 included *normal* census SoundEx codes as well as these **special** codes randomly intermixed. For this reason you must search these years using SoundEx indexes made in both ways.

For our SoundEx function code, we will use a parameter called **CensusOption**. We will call the ***normal*** method of calculating the census SoundEx method 1. We will call the incorrect ***special*** method sometimes used in pre-1920 censuses method 2. If we don't need census compatible codes, we can instruct our function to generate codes using our enhancements by setting CensusOption to 0 (or just not including it at all in languages that permit). Here it is in a table.

CensusOption	SoundEx Code Returned
0	Not census codes Enhanced SoundEx as documented here
1	Normal census codes Used in all censuses including 1920
2	Special census codes Used intermittently in 1880,* 1900, 1910 censuses

**1890 census records were destroyed in a fire.*

Also of importance for those searching through census records: Often census takers would leave off the prefix in names that had prefixes when producing SoundEx codes (nobody's really sure why). In this case the SoundEx for the name VanDrake (for example) might have the soundex calculated only for 'Drake'.

One more thing - Out on the web, some are saying the incorrect "special" census codes documented here are actually the correctly done census codes, while those that followed the special rule for 'H' and 'W' were the ones done in error. They are wrong. Fortunately though, you don't have to decide which side to believe (but, still, have I ever lied to you? :-)).

Simple logic should provide enough evidence to allow you to draw your own conclusion. Consider...

- In order to accept the argument that: 'codes that lack a special rule for 'H' and 'W' are the "correct" codes', you would have to believe that those census takers who "mistakenly" followed the special rule for 'H' and 'W' all spontaneously ***made up*** the exact same ***non-existent*** rule for 'H' and 'W'.

  In other words, with such an argument, you're not asked to believe they accidentally ***ignored*** a special rule for 'H' and 'W'. Instead, you're asked to believe they all *independently* ***created*** the exact same errant rule out of whole-cloth, and then followed it. That's many different individuals in different states...

If logic makes your eyes glaze over, don't worry. A very convincing appeal to authority exists: No less than the U.S. National Archives (those who maintain census data) agree with the position documented here. :-) See the Related Reading section below for a link.

[top]

SoundEx Limitations

SoundEx acts as a bridge between the fuzzy and inexact process of human vocal interaction, and the concise true/false processes at the foundation of computer communication. As such, SoundEx is an inherently unreliable interface.

For this reason, SoundEx is only usable in applications that can tolerate high false positives (when words that don't match the sound of the inquiry are returned) and high false negatives (when words that match the sound of the inquiry are NOT returned).

This limitation is true even of the best SoundEx improvement techniques available. As long as you accept and honor this limitation, SoundEx and its derivatives can be a very useful tool in helping to improve the quality and usefulness of databases.

In many instances, unreliable interfaces are used as a foundation, upon which a reliable layer may be built. Interfaces that build a reliable layer, based on context, over a SoundEx foundation may also be possible.

[top]

Permissions

[top]

Source Code

Below you will find a set of SoundEx functions that produce identical results based on identical inputs. They are currently provided in the following languages:

C Code	JavaScript	Perl	VB

These functions are provided under ***Berkeley*** styled licensing (included above each function). If you'd like to contribute this function here in your own favorite language under your own similar licensing, please let me know.

The function they perform will return SoundEx codes as produced in the census, or enhanced by techniques documented in this article. The JavaScript version of the function is identical to the one used in the example SoundEx converter form provided below.

Note on internationalization:
These functions are only for words that use characters from the first seven bits of utf-8 (traditionally called 7-bit ASCII).

Beware of SoundEx functions on the web that claim to be international. Some will claim, and truly believe, that their code is international simply because they've used the standard (internationalized) library functions to handle strings. This assumption is often simply a mistake made by a real programmer who is an El Niño (had to stretch a little for the pun there :-) sorry).

SoundEx in **C**

```
/*
 * v 1.0d  TESTED-OK  20060308
 * -----------------------
 *
 * The following SoundEx function is:
 *
 *    (C) Copyright 2002 - 2006, Creativyst, Inc.
 *              ALL RIGHTS RESERVED
 *
 * For more information go to:
 *          http://www.Creativyst.com
 * or email:
 *          Support@Creativyst.com
 *
 * Redistribution and use in source and binary
 * forms, with or without modification, are
 * permitted provided that the following conditions
 * are met:
 *
 *   1. Redistributions of source code must
 *      retain the above copyright notice, this
 *      list of conditions and the following
 *      disclaimer.
 *
 *   2. Redistributions in binary form must
 *      reproduce the above copyright notice,
 *      this list of conditions and the
 *      following disclaimer in the
 *      documentation and/or other materials
 *      provided with the distribution.
 *
 *   3. All advertising materials mentioning
 *      features or use of this software must
 *      display the following acknowledgement:
 *      This product includes software developed
 *      by Creativyst, Inc.
 *
 *   4. The name of Creativyst, Inc. may not be
 *      used to endorse or promote products
 *      derived from this software without
 *      specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY CREATIVYST CORPORATION
 *`AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
 * THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
```

```
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *
 * ------------------
 * ------------------
 * FUNCTION NOTES:
 *  1. To avoid all possibility of overwrites make
 *     sure *SoundEx points to a buffer with at least
 *     11 bytes of storage.
 *
 *  2. This function is for 7/8-bit ASCII characters.
 *     Modifications are required for UTF16/32, or for
 *     anything other than the first 7-bits of utf-8.
 *
 *  3. For those embedded guys who will understand this:
 *     This is a true library-grade (i.e. re-usable) function,
 *     meaning it has no dependencies on outside functions
 *     and requires no non-standard libraries be linked in
 *     order for it to work. In this case, since it doesn't
 *     even require the standard C library, it is what C99
 *     (I think) calls a: strictly conforming freestanding
 *     function.
 *
*/
 int SoundEx(char *SoundEx,
             char *WordString,
             int   LengthOption,
             int   CensusOption)
 {
     int  InSz = 31;
     char WordStr[32];     /* one bigger than InSz */
     int  SoundExLen, WSLen, i;
     char FirstLetter, *p, *p2;

     SoundExLen = WSLen = 0;
     SoundEx[0] = 0;

     if(CensusOption) {
       LengthOption = 4;
     }

     if(LengthOption) {
         SoundExLen = LengthOption;
     }
     if(SoundExLen > 10) {
         SoundExLen = 10;
     }
     if(SoundExLen < 4) {
         SoundExLen = 4;
     }
```

```
    if(!WordString) {
        return(0);
    }

    /* Copy WordString to WordStr
     * without using funcs from other
     * libraries.
    */
    for(p = WordString,p2 = WordStr,i = 0;(*p);p++,p2++,i++) {
      if(i >= InSz) break;
      (*p2) = (*p);
    }
    (*p2) = 0;



    /* Convert WordStr to
     * upper-case, without using funcs
     * from other libraries
    */
    for(p = WordStr;(*p);p++) {
      if( (*p) >= 'a' && (*p) <= 'z' ) {
          (*p) -= 0x20;
      }
    }


    /* convert all non-alpha
     * chars to spaces
    */
    for(p = WordStr;(*p);p++) {
      if( (*p) < 'A' || (*p) > 'Z' ) {
          (*p) = ' ';
      }
    }

    /* Remove leading spaces
    */
    for(i = 0, p = p2 = WordStr;(*p);p++) {
        if(!i) {
            if( (*p) != ' ' ) {
                (*p2) = (*p);
                p2++;
                i++;
            }
        }
        else {
            (*p2) = (*p);
            p2++;
        }
    }
    (*p2) = 0;

    /* Get length of WordStr
    */
    for(i = 0,p = WordStr;(*p);p++) i++;
```

```
    /* Remove trailing spaces
    */
    for(;i;i--) {
        if(WordStr[i] == ' ') {
          WordStr[i] = 0;
        }
        else {
            break;
        }
    }

    /* Get length of WordStr
    */
    for(WSLen = 0,p = WordStr;(*p);p++) WSLen++;

    if(!WSLen) {
      return(0);
    }



    /* Perform our own multi-letter
     * improvements
     *
     * underscore placeholders (_) will be
     * removed below.
    */
    if(!CensusOption) {
        if(WordStr[0] == 'P' && WordStr[1] == 'S') {
            WordStr[0] = '_';
        }
        if(WordStr[0] == 'P' && WordStr[1] == 'F') {
            WordStr[0] = '_';
        }

        for(i = 0;i < WSLen;i++) {
            if(WordStr[i] == 'D' && WordStr[i+1] == 'G') {
                WordStr[i] = '_';
                i++;
                continue;
            }
            if(WordStr[i] == 'G' && WordStr[i+1] == 'H') {
                WordStr[i] = '_';
                i++;
                continue;
            }
            if(WordStr[i] == 'K' && WordStr[i+1] == 'N') {
                WordStr[i] = '_';
                i++;
                continue;
            }
            if(WordStr[i] == 'G' && WordStr[i+1] == 'N') {
                WordStr[i] = '_';
                i++;
                continue;
```

```
            }
            if(WordStr[i] == 'M' && WordStr[i+1] == 'B') {
                WordStr[i+1] = '_';
                i++;
                continue;
            }

            if(WordStr[i] == 'P' && WordStr[i+1] == 'H') {
                WordStr[i] = 'F';
                WordStr[i+1] = '_';
                i++;
                continue;
            }
            if(WordStr[i]  ==  'T'  &&
               WordStr[i+1] == 'C' &&
               WordStr[i+2] == 'H'
              ) {

                WordStr[i] = '_';
                i++; i++;
                continue;
            }
            if(WordStr[i] == 'M' && WordStr[i+1] == 'P'
               && (WordStr[i+2] == 'S' ||
                   WordStr[i+2] == 'T' ||
                   WordStr[i+2] == 'Z')
              ) {
                WordStr[i+1] = '_';
                i++;
            }
        }
    } /* end if(!CensusOption) */



    /* squeeze out underscore characters
     * added as a byproduct of above process
     * (only needed in c styled replace)
    */
    for(p = p2 = WordStr;(*p);p++) {
      (*p2) = (*p);
      if( (*p2) != '_' ) {
          p2++;
      }
    }
    (*p2) = 0;




    /* This must be done AFTER our
     * multi-letter replacements
     * since they could change
     * the first letter
    */
    FirstLetter = WordStr[0];
```

```
/* In case we're in CensusOption
 * 1 and the word starts with
 * an 'H' or 'W'
 *  (v1.0c djr: add test for H or W)
*/
if(FirstLetter == 'H' || FirstLetter == 'W') {
    WordStr[0] = '-';
}



/* In properly done census
 * SoundEx, the H and W will
 * be squezed out before
 * performing the test
 * for adjacent digits
 * (this differs from how
 * 'real' vowels are handled)
*/
if(CensusOption == 1) {
    for(p = &(WordStr[1]);(*p);p++) {
       if((*p) == 'H' || (*p) == 'W') {
           (*p) = '.';
       }
    }
}




/* Perform classic SoundEx
 * replacements.
*/
for(p = WordStr;(*p);p++) {
    if( (*p) == 'A'   ||
        (*p) == 'E'   ||
        (*p) == 'I'   ||
        (*p) == 'O'   ||
        (*p) == 'U'   ||
        (*p) == 'Y'   ||
        (*p) == 'H'   ||
        (*p) == 'W'
      ){
        (*p) = '0';   /* zero */
    }
    if( (*p) == 'B'   ||
        (*p) == 'P'   ||
        (*p) == 'F'   ||
        (*p) == 'V'
      ){
        (*p) = '1';
    }
    if( (*p) == 'C'   ||
        (*p) == 'S'   ||
```

```
            (*p) == 'G'   ||
            (*p) == 'J'   ||
            (*p) == 'K'   ||
            (*p) == 'Q'   ||
            (*p) == 'X'   ||
            (*p) == 'Z'
          ){
            (*p) = '2';
        }
        if( (*p) == 'D'   ||
            (*p) == 'T'
          ){
            (*p) = '3';
        }
        if( (*p) == 'L' ) {
            (*p) = '4';
        }

        if( (*p) == 'M'   ||
            (*p) == 'N'
          ){
            (*p) = '5';
        }
        if( (*p) == 'R' ) {
            (*p) = '6';
        }
    }
    /* soundex replacement loop done  */




    /* In properly done census
     * SoundEx, the H and W will
     * be squezed out before
     * performing the test
     * for adjacent digits
     * (this differs from how
     * 'real' vowels are handled)
    */
    if(CensusOption == 1) {
        /* squeeze out dots
        */
        for(p = p2 = &WordStr[1];(*p);p++) {
          (*p2) = (*p);
          if( (*p2) != '.' ) {
              p2++;
          }
        }
        (*p2) = 0;
    }



    /* squeeze out extra equal adjacent digits
     * (don't include first letter)
     * v1.0c djr (now includes first letter)
```

```
    */
    for(p = p2 = &(WordStr[0]);(*p);p++) {
      (*p2) = (*p);
      if( (*p2) != p[1] ) {
          p2++;
      }
    }
    (*p2) = 0;



    /* squeeze out spaces and zeros
     * Leave the first letter code
     * to be replaced below.
     * (In case it made a zero)
    */
    for(p = p2 = &WordStr[1];(*p);p++) {
      (*p2) = (*p);
      if( (*p2) != ' ' && (*p2) != '0' ) {
          p2++;
      }
    }
    (*p2) = 0;



    /* Get length of WordStr
    */
    for(WSLen = 0,p = WordStr;(*p);p++) WSLen++;


    /* Right pad with zero characters
    */
    for(i = WSLen;i < SoundExLen;i++ ) {
        WordStr[i] = '0';
    }

    /* Size to taste
    */
    WordStr[SoundExLen] = 0;


    /* Replace first digit with
     * first letter.
    */
    WordStr[0] = FirstLetter;


    /* Copy WordStr to SoundEx
    */
    for(p2 = SoundEx,p = WordStr;(*p);p++,p2++) {
        (*p2) = (*p);
    }
    (*p2) = 0;

    return(SoundExLen);
}
```

[top]

SoundEx in **JavaScript**

```
/*
 * v 1.0d  TESTED-OK  20060112
 * -----------------------
 *
 * The following SoundEx function is:
 *
 *    (C) Copyright 2002 - 2006, Creativyst, Inc.
 *               ALL RIGHTS RESERVED
 *
 * For more information go to:
 *           http://www.Creativyst.com
 * or email:
 *           Support@Creativyst.com
 *
 * Redistribution and use in source and binary
 * forms, with or without modification, are
 * permitted provided that the following conditions
 * are met:
 *
 *   1. Redistributions of source code must
 *      retain the above copyright notice, this
 *      list of conditions and the following
 *      disclaimer.
 *
 *   2. Redistributions in binary form must
 *      reproduce the above copyright notice,
 *      this list of conditions and the
 *      following disclaimer in the
 *      documentation and/or other materials
 *      provided with the distribution.
 *
 *   3. All advertising materials mentioning
 *      features or use of this software must
 *      display the following acknowledgement:
 *      This product includes software developed
 *      by Creativyst, Inc.
 *
 *   4. The name of Creativyst, Inc. may not be
 *      used to endorse or promote products
 *      derived from this software without
 *      specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY CREATIVYST CORPORATION
```

```
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
 * THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
*/
 function SoundEx(WordString, LengthOption, CensusOption)
 {
     var TmpStr;
     var WordStr = "";
     var CurChar;
     var LastChar;
     var SoundExLen = 10;
     var WSLen;
     var FirstLetter;

     if(CensusOption) {
         LengthOption = 4;
     }

     if(LengthOption != undefined) {
         SoundExLen = LengthOption;
     }
     if(SoundExLen > 10) {
         SoundExLen = 10;
     }
     if(SoundExLen < 4) {
         SoundExLen = 4;
     }

     if(!WordString) {
         return("");
     }

     WordString = WordString.toUpperCase();

     /* Clean and tidy
     */
     WordStr = WordString;


     WordStr = WordStr.replace(/[^A-Z]/gi, " "); // rpl non-chars w space
     WordStr = WordStr.replace(/^\s*/g, "");     // remove leading space
     WordStr = WordStr.replace(/\s*$/g, "");     // remove trailing space
```

```
    if(!CensusOption) {
        /* Some of our own improvements
        */
        WordStr = WordStr.replace(/DG/g, "G");     // Change DG to G
        WordStr = WordStr.replace(/GH/g, "H");     // Change GH to H
        WordStr = WordStr.replace(/GN/g, "N");     // Change GN to N
        WordStr = WordStr.replace(/KN/g, "N");     // Change KN to N
        WordStr = WordStr.replace(/PH/g, "F");     // Change PH to F
        WordStr =
            WordStr.replace(/MP([STZ])/g, "M$1"); // MP if fllwd by ST|Z
        WordStr = WordStr.replace(/^PS/g, "S");   // Chng leadng PS to S
        WordStr = WordStr.replace(/^PF/g, "F");   // Chng leadng PF to F
        WordStr = WordStr.replace(/MB/g, "M");    // Chng MB to M
        WordStr = WordStr.replace(/TCH/g, "CH");  // Chng TCH to CH
    }




    /* The above improvements
     * may change this first letter
    */
    FirstLetter = WordStr.substr(0,1);


    /* in case 1st letter is
     * an H or W and we're in
     * CensusOption = 1
    */
    if(FirstLetter == "H" || FirstLetter == "W") {
        TmpStr = WordStr.substr(1);
        WordStr = "-";
        WordStr += TmpStr;
    }


    /* In properly done census
     * SoundEx the H and W will
     * be squezed out before
     * performing the test
     * for adjacent digits
     * (this differs from how
     * 'real' vowels are handled)
    */
    if(CensusOption == 1) {
        WordStr = WordStr.replace(/[HW]/g, ".");
    }


    /* Begin Classic SoundEx
    */
    WordStr =  WordStr.replace(/[AEIOUYHW]/g, "0");
    WordStr = WordStr.replace(/[BPFV]/g, "1");
    WordStr = WordStr.replace(/[CSGJKQXZ]/g, "2");
    WordStr = WordStr.replace(/[DT]/g, "3");
    WordStr = WordStr.replace(/[L]/g, "4");
```

```
    WordStr = WordStr.replace(/[MN]/g, "5");
    WordStr = WordStr.replace(/[R]/g, "6");



    /* Properly done census:
     * squeze H and W out
     * before doing adjacent
     * digit removal.
    */
    if(CensusOption == 1) {
        WordStr = WordStr.replace(/\./g, "");
    }



    /* Remove extra equal adjacent digits
    */
    WSLen = WordStr.length;
    LastChar = "";
    TmpStr = "";
    // removed v10c djr:  TmpStr = "-";  /* rplcng skipped first char */

    for(i = 0; i < WSLen; i++) {
        CurChar = WordStr.charAt(i);
        if(CurChar == LastChar) {
            TmpStr += " ";
        }
        else {
            TmpStr += CurChar;
            LastChar = CurChar;
        }
    }
    WordStr = TmpStr;


    WordStr = WordStr.substr(1);          /* Drop first letter code   */
    WordStr = WordStr.replace(/\s/g, ""); /* remove spaces            */
    WordStr = WordStr.replace(/0/g, "");  /* remove zeros             */
    WordStr += "0000000000";              /* pad with zeros on right  */

    WordStr = FirstLetter + WordStr;      /* Add first letter of word */

    WordStr = WordStr.substr(0,SoundExLen); /* size to taste     */

    return(WordStr);
}
```

[top]

SoundEx in **Perl**

```
# v 1.0d  TESTED-OK  20061006
# -----------------------
# The following SoundEx function is:
#
#    (C) Copyright 2002 - 2006, Creativyst, Inc.
#                ALL RIGHTS RESERVED
#
# For more information go to:
#           http://www.Creativyst.com
# or email:
#           Support@Creativyst.com
#
# Redistribution and use in source and binary
# forms, with or without modification, are
# permitted provided that the following conditions
# are met:
#
#   1. Redistributions of source code must
#      retain the above copyright notice, this
#      list of conditions and the following
#      disclaimer.
#
#   2. Redistributions in binary form must
#      reproduce the above copyright notice,
#      this list of conditions and the
#      following disclaimer in the
#      documentation and/or other materials
#      provided with the distribution.
#
#   3. All advertising materials mentioning
#      features or use of this software must
#      display the following acknowledgement:
#      This product includes software developed
#      by Creativyst, Inc.
#
#   4. The name of Creativyst, Inc. may not be
#      used to endorse or promote products
#      derived from this software without
#      specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY CREATIVYST CORPORATION
# ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,
# INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
# WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
# PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
# THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO,
# PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
# OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
# WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
# WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
```

```
# ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
#
sub SoundEx
{
    my($WordString, $LengthOption, $CensusOption) = @_;
    my($WordStr, $CurChar, $LastChar, $SoundExLen);
    my($WSLen, $FirstLetter, $TmpStr);

    if($CensusOption) {
        $LengthOption = 4;
    }

    if($LengthOption) {
        $SoundExLen = $LengthOption;
    }
    if($SoundExLen > 10) {
        $SoundExLen = 10;
    }
    if($SoundExLen < 4) {
        $SoundExLen = 4;
    }

    if(!$WordString) {
        return("");
    }

    $WordString = uc($WordString);
    # Clean and tidy
    #
    $WordStr = $WordString;
    $WordStr =~ s/[^A-Z]/ /sig;   # replace non-chars with space
    $WordStr =~ s/^\s*//sg;       # remove leading space
    $WordStr =~ s/\s*$//sg;       # remove trailing space


    if(!$CensusOption) {
        # Some of our own improvements
        #
        $WordStr =~ s/DG/G/sg;          # Change DG to G
        $WordStr =~ s/GH/H/sg;          # Change GH to H
        $WordStr =~ s/KN/N/sg;          # Change KN to N
        $WordStr =~ s/GN/N/sg;          # Change GN to N
        $WordStr =~ s/MB/M/sg;          # Change MB to M
        $WordStr =~ s/PH/F/sg;          # Change PH to F
        $WordStr =~ s/TCH/CH/sg;        # Change TCH to CH
        $WordStr =~ s/MP([STZ])/M$1/sg; # MP if follwd by S|T|Z
        $WordStr =~ s/^PS/S/sg;         # Change leading PS to S
        $WordStr =~ s/^PF/F/sg;         # Change leading PF to F
    }


    # Done here because the
    # above improvements could
    # change this first letter
    #
    $FirstLetter = substr($WordStr,0,1);
```

```
# in case 1st letter is
# an H or W and we're in
# CensusOption = 1
#  (add test for 'H'/'W' v1.0c djr)
#
if($FirstLetter eq "H" || $FirstLetter eq "W") {
    $TmpStr = substr($WordStr,1);
    $WordStr = "-$TmpStr";
}



# In properly done census
# SoundEx: the H and W will
# be squeezed out before
# performing the test for
# adjacent digits
# (this differs from how
# the 'real' vowels are
# handled)
#
if($CensusOption == 1) {
    $WordStr =~ s/[HW]/\./sg;
}


# Begin Classic SoundEx
#
$WordStr =~ s/[AEIOUYHW]/0/sg;
$WordStr =~ s/[BPFV]/1/sg;
$WordStr =~ s/[CSGJKQXZ]/2/sg;
$WordStr =~ s/[DT]/3/sg;
$WordStr =~ s/L/4/sg;
$WordStr =~ s/[MN]/5/sg;
$WordStr =~ s/R/6/sg;


# Properly done census:
# squeeze H and W out
# before doing adjacent
# digit removal.
#
if($CensusOption == 1) {
    $WordStr =~ s/\.//sg;
}



# Remove extra equal adjacent digits
#
$WSLen = length($WordStr);
$LastChar = "";
# v1.0c rmv: $TmpStr = "-";    # rplc skipped 1st char
$TmpStr = "";
for($i = 0; $i < $WSLen;$i++) { # v1.0c now org-0
```

```
        $CurChar = substr($WordStr,$i,1);
        if($CurChar eq $LastChar) {
            $TmpStr .= " ";
        }
        else {
            $TmpStr .= $CurChar;
            $LastChar = $CurChar;
        }
    }
    $WordStr = $TmpStr;


    $WordStr = substr($WordStr,1);      # Drop first ltr code
    $WordStr =~ s/\s//sg;               # remove spaces
    $WordStr =~ s/0//sg;                # remove zeros
    $WordStr .= "0000000000";           # pad w/0s on rght

    $WordStr = "$FirstLetter$WordStr";  # Add 1st ltr of wrd

    $WordStr = substr($WordStr,0,$SoundExLen);  # size to taste

    return($WordStr);
}
```

[top]

SoundEx in **VB**

```
'
' v 1.0d  TESTED-OK  20061009
' ----------------------
'
' The following SoundEx function is:
'
'    (C) Copyright 2002 - 2006, Creativyst, Inc.
'                ALL RIGHTS RESERVED
'
' For more information go to:
'           http://www.Creativyst.com
' or email:
'           Support@Creativyst.com
'
' Redistribution and use in source and binary
' forms, with or without modification, are
' permitted provided that the following conditions
' are met:
'
'   1. Redistributions of source code must
'      retain the above copyright notice, this
'      list of conditions and the following
```

```
'      disclaimer.
'
'   2. Redistributions in binary form must
'      reproduce the above copyright notice,
'      this list of conditions and the
'      following disclaimer in the
'      documentation and/or other materials
'      provided with the distribution.
'
'   3. All advertising materials mentioning
'      features or use of this software must
'      display the following acknowledgement:
'      This product includes software developed
'      by Creativyst, Inc.
'
'   4. The name of Creativyst, Inc. may not be
'      used to endorse or promote products
'      derived from this software without
'      specific prior written permission.
'
' THIS SOFTWARE IS PROVIDED BY CREATIVYST CORPORATION
' ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,
' INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
' WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
' PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
' THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
' INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
' DAMAGES (INCLUDING, BUT NOT LIMITED TO,
' PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
' OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
' HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
' WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
' (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
' WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
' ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
'
'
'
Function SoundEx( _
                ByVal WordString As String, _
                ByVal LengthOption As Integer, _
                ByVal CensusOption As Integer _
                ) As String

Dim WordStr As String
Dim b, b2, b3, SoundExLen, FirstLetter As String
Dim i As Integer


' Sanity
'
If (CensusOption > 0) Then
    LengthOption = 4
End If
If (LengthOption > 0) Then
    SoundExLen = LengthOption
End If
```

```
If (SoundExLen > 10) Then
    SoundExLen = 10
End If
If (SoundExLen < 4) Then
    SoundExLen = 4
End If
If (Len(WordString) < 1) Then
    Exit Function
End If


' Copy to WordStr
' and UpperCase
'
WordStr = UCase(WordString)


' Convert all non-alpha
' chars to spaces. (thanks John)
'
For i = 1 To Len(WordStr)
    b = Mid(WordStr, i, 1)
    If (Not (b Like "[A-Z]")) Then
        WordStr = Replace(WordStr, b, " ")
    End If
Next i


' Remove leading and
' trailing spaces
'
WordStr = Trim(WordStr)

' sanity
'
If (Len(WordStr) < 1) Then
    Exit Function
End If


' Perform our own multi-letter
' improvements
'
' double letters will be effectively
' removed in a later step.
'
If (CensusOption < 1) Then
    b = Mid(WordStr, 1, 1)
    b2 = Mid(WordStr, 2, 1)
    If (b = "P" And b2 = "S") Then
        WordStr = Replace(WordStr, "PS", "S", 1, 1)
    End If
    If (b = "P" And b2 = "F") Then
        WordStr = Replace(WordStr, "PF", "F", 1, 1)
    End If

    WordStr = Replace(WordStr, "DG", "_G")
```

```
    WordStr = Replace(WordStr, "GH", "_H")
    WordStr = Replace(WordStr, "KN", "_N")
    WordStr = Replace(WordStr, "GN", "_N")
    WordStr = Replace(WordStr, "MB", "M_")
    WordStr = Replace(WordStr, "PH", "F_")
    WordStr = Replace(WordStr, "TCH", "_CH")
    WordStr = Replace(WordStr, "MPS", "M_S")
    WordStr = Replace(WordStr, "MPT", "M_T")
    WordStr = Replace(WordStr, "MPZ", "M_Z")

End If
' end if(Not CensusOption)
'

' Sqeeze out the extra _ letters
' from above (not strictly needed
' in VB but used in C code)
'
WordStr = Replace(WordStr, "_", "")



' This must be done AFTER our
' multi-letter replacements
' since they could change
' the first letter
'
FirstLetter = Mid(WordStr, 1, 1)

' in case first letter is
' an h, a w ...
' we'll change it to something
' that doesn't match anything
'
If (FirstLetter = "H" Or FirstLetter = "W") Then
    b = Mid(WordStr, 2)
    WordStr = "-" + b
End If


' In properly done census
' SoundEx, the H and W will
' be squezed out before
' performing the test
' for adjacent digits
' (this differs from how
' 'real' vowels are handled)
'
If (CensusOption = 1) Then
    WordStr = Replace(WordStr, "H", ".")
    WordStr = Replace(WordStr, "W", ".")
End If


' Perform classic SoundEx
' replacements
' Here, we use ';' instead of zero '0'
```

```
' because of MS strangeness with leading
' zeros in some applications.
'
WordStr = Replace(WordStr, "A", ";")
WordStr = Replace(WordStr, "E", ";")
WordStr = Replace(WordStr, "I", ";")
WordStr = Replace(WordStr, "O", ";")
WordStr = Replace(WordStr, "U", ";")
WordStr = Replace(WordStr, "Y", ";")
WordStr = Replace(WordStr, "H", ";")
WordStr = Replace(WordStr, "W", ";")

WordStr = Replace(WordStr, "B", "1")
WordStr = Replace(WordStr, "P", "1")
WordStr = Replace(WordStr, "F", "1")
WordStr = Replace(WordStr, "V", "1")

WordStr = Replace(WordStr, "C", "2")
WordStr = Replace(WordStr, "S", "2")
WordStr = Replace(WordStr, "G", "2")
WordStr = Replace(WordStr, "J", "2")
WordStr = Replace(WordStr, "K", "2")
WordStr = Replace(WordStr, "Q", "2")
WordStr = Replace(WordStr, "X", "2")
WordStr = Replace(WordStr, "Z", "2")

WordStr = Replace(WordStr, "D", "3")
WordStr = Replace(WordStr, "T", "3")

WordStr = Replace(WordStr, "L", "4")

WordStr = Replace(WordStr, "M", "5")
WordStr = Replace(WordStr, "N", "5")

WordStr = Replace(WordStr, "R", "6")
'
' End Clasic SoundEx replacements
'



' In properly done census
' SoundEx, the H and W will
' be squezed out before
' performing the test
' for adjacent digits
' (this differs from how
' 'real' vowels are handled)
'
If (CensusOption = 1) Then
    WordStr = Replace(WordStr, ".", "")
End If



' squeeze out extra equal adjacent digits
' (don't include first letter)
```

```
'
b = ""
b2 = ""
' remove from v1.0c djr: b3 = Mid(WordStr, 1, 1)
b3 = ""
For i = 1 To Len(WordStr) ' i=1 (not 2) in v1.0c
    b = Mid(WordStr, i, 1)
    b2 = Mid(WordStr, (i + 1), 1)
    If (Not (b = b2)) Then
        b3 = b3 + b
    End If
Next i

WordStr = b3
If (Len(WordStr) < 1) Then
    Exit Function
End If



' squeeze out spaces and zeros (;)
' Leave the first letter code
' to be replaced below.
' (In case it made a zero)
'
WordStr = Replace(WordStr, " ", "")
b = Mid(WordStr, 1, 1)
WordStr = Replace(WordStr, ";", "")
If (b = ";") Then ' only if it got removed above
    WordStr = b + WordStr
End If



' Right pad with zero characters
'
b = String(SoundExLen, "0")
WordStr = WordStr + b


' Replace first digit with
' first letter
'
WordStr = Mid(WordStr, 2)
WordStr = FirstLetter + WordStr

' Size to taste
'
WordStr = Mid(WordStr, 1, SoundExLen)


' Copy WordStr to SoundEx
'
SoundEx = WordStr

End Function
```

[top]

Related Reading

The following links provide resources for those who'd like to learn more about the SoundEx algorithm, it's limitations, enhancements, and uses.

- Surname to Soundex Converter
- Explanation of census SoundEx at National Archives (U.S.)
- PDF: Improving Precision and Recall for SoundEx Retrieval
- PDF: Computational Techniques for Improving Name Search

[top]

SoundEx Converter Form

The following form uses the JavaScript SoundEx function presented above to produce a SoundEx code for any word you enter into the field labeled "Input Word:". It returns the SoundEx codes for three different algorithms in the fields below. You may also enter a size for the output code within limits (4 to 10 characters).

Creativyst® SoundExR
+Add to your site Pop-up

Input Word: go

Size: 4

0: Enhanced SoundEx:
As documented in this article about SoundEx

1: Normal Census
Properly calculated SoundEx codes found in all census years.

2: Special Census
Improperly calculated SoundEx codes found in SOME of the censuses performed in 1880, 1900, and 1910.

To test differences in the enhanced SoundEx option, use words and names like *knightridder*, *psychology*, and *Pflanders*. The word *Knight* will demonstrate two different enhancements as you type it. To see the differences between the two census options, try *Ashcroft*.

[top]

Contribute

If you've written a SoundEx function in another language and would like to share it as part of this article, please contact me. If used, your code will be attributed to you with a link to your site.

Written by Dominic John Repici

Thanks for your help & contributions:
John Nye (Tool guy) for finding bugs in Perl code
Dmitry S. Denisov for bug-finding (thanks!)
Mark Westenberger (mjw -at: igs.net) for finding bugs in C code

```
tests:

   v1.0d:
     Include test-strings with non-alpha characters.

   v1.0c:
     Schmit      S530 (not S253)
     Schneider   S536 (not S253)
```

```
    Lloyd       L300 (not L430)
    Pfister     P236

  v1.0b:
    Holden      (first letter is 'H')
    Write       (first letter is 'W' followed by #6 letter)
    White       (oooh, H/W together in 1st letter)
    Htacky      H320
    Atacky      A320
    ashcroft    (to test normal H squeezes)
```