Version 1.0
April 18, 2008

How do I... Implement the Soundex function in C#?

By Zach Smith

For years Microsoft SQL Server has provided developers with a method called Soundex that is used to retrieve an encoded string. Words that sound alike have similar encodings, so you can use this functionality to provide some flexibility in searches. This How do I... blog post shows application developers how to implement Soundex completely in C# without having to use the SQL function.

Soundex introduction

Soundex is an algorithm that converts words to an encoded string based on the way the word is pronounced. This allows you to compare words based on pronunciation instead of binary matches. For example, Zach and Zack are pronounced exactly the same way. However, the string "Zach" does not equal the string "Zack" which means that a normal search would not mark them as a match. After running "Zach" and "Zack" through Soundex you will see that they both have the same encoding:

- Zach encodes as "Z200"
- Zack also encodes as "Z200"

This works by breaking the string down into individual characters and assigning each character a value. The rules for the algorithm are shown below:

1. The first letter of the encoding will always be the first letter of the string
2. Ignore the letters a, e, h, I, o, u, w, and y except in rule #1
3. Use the table below to assign the remaining letters their encoding:
    a. 1 should be assigned to b, f, p, and v
    b. 2 should be assigned to c, g, j, k, q, s, x, and z
    c. 3 should be assigned to d and t
    d. 4 should be assigned to l
    e. 5 should be assigned to m and n
    f. 6 should be assigned to r
4. Any letters with the same encoding that appear subsequently to each other should be ignored. For instance the sequence "BB" would produce the encoding "1". The sequence "BAB" would also produce the encoding "1" since A is an ignored character.
5. The total length of the encoding must always be four. If you have more than four characters in your encoding, trim the extra characters. If you have less than four characters pad the encoding with zeroes to bring the length to four.

Below are a few examples produced using this algorithm:

- Donald – D543
- Zach – Z200
- Campbel – C514
- Cammmppppbbbeeelll – C514
- David – D130

Another thing to note is that Soundex is case insensitive. So "ZACH" and "zAcH" both produce the same encoding.

Implementing Soundex in C#

The code shown in **Figure A** demonstrates one way to implement Soundex in C#. While there are surely other methods, this one seems to be a good balance between readability and performance.

Figure A

```
public static string Soundex(string data)
{
    StringBuilder result = new StringBuilder();

    if (data != null && data.Length > 0)
    {
        string previousCode = "", currentCode = "",
                currentLetter = "";

        result.Append(data.Substring(0, 1));

        for (int i = 1; i < data.Length; i++)
        {
            currentLetter = data.Substring(i, 1).ToLower();
            currentCode = "";

            if ("bfpv".IndexOf(currentLetter) > -1)
                currentCode = "1";
            else if ("cgjkqsxz".IndexOf(currentLetter) > -1)
                currentCode = "2";
            else if ("dt".IndexOf(currentLetter) > -1)
                currentCode = "3";
            else if (currentLetter == "l")
                currentCode = "4";
            else if ("mn".IndexOf(currentLetter) > -1)
                currentCode = "5";
            else if (currentLetter == "r")
                currentCode = "6";

            if (currentCode != previousCode)
                result.Append(currentCode);

            if (result.Length == 4) break;

            if(currentCode != "")
                previousCode = currentCode;
        }
    }

    if (result.Length < 4)
        result.Append(new String('0', 4 - result.Length));

    return result.ToString().ToUpper();
}
```

The Soundex Function

The above code loops through the data supplied and determines which encoding, if any, should be applied to the current character. It then pads the encoding with zeroes if necessary and returns the encoding.

Calculating the difference between Soundex codes

There are times when similar words do not have the same Soundex encoding. Take, for example, the words "Lake" and "Bake". These words sound very similar but the encoding is not the exactly the same. Because of this, Microsoft SQL Server includes a function called "Difference" that will give you a ranking on how similar two words are. The rank is from four to one, with four being a perfect match and one representing no match at all.

Unlike the Soundex algorithm, the Difference function does not use a published formula to determine the ranking. These are rules I have developed, and they seem to closely match the rules used by the Difference function in SQL:

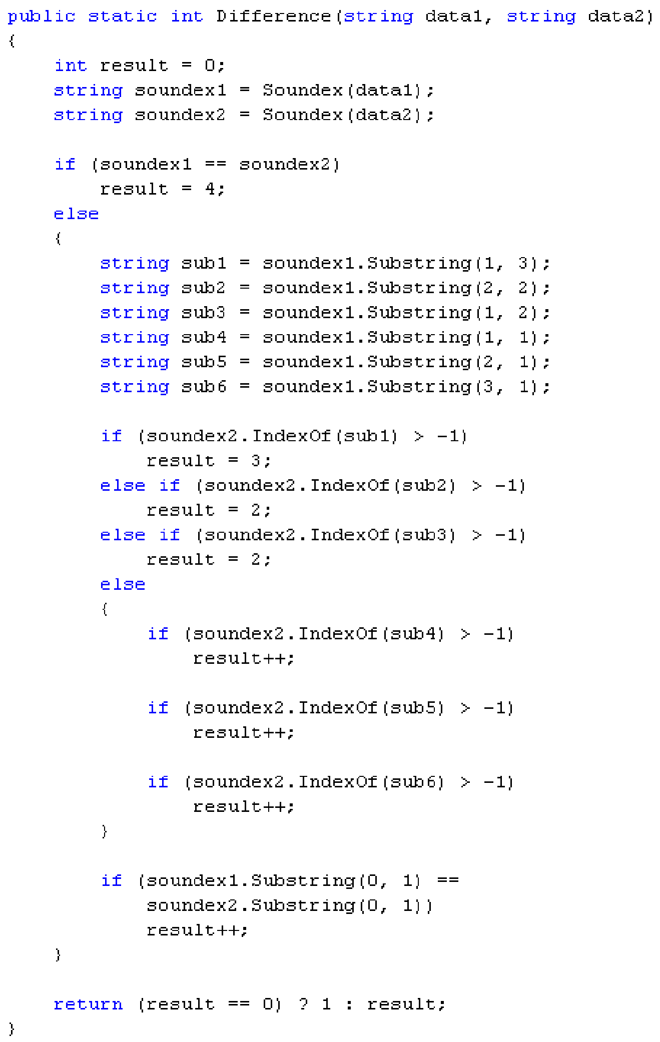
1. Run both words through Soundex
2. If the Soundex encodings are the same the rank will be 4. If the Soundex encodings are not the same, continue to step 3.
3. If the last three characters of the first encoding are found in the second encoding the rank will be 3. Skip to step 7.
4. If the last two characters of the first encoding are found in the second encoding the rank will be 2. Skip to step 7.
5. If the middle two characters of the first encoding are found in the second encoding, the rank will be 2. Skip to step 7.
6. If the second, third, or fourth characters of the first encoding are found within the second encoding add 1 to the rank for each character that matches.
7. If the first character of the first encoding matches the first character of the second encoding, add 1 to the current rank.

These rules are approximate, but they produce very similar results to the SQL Difference function. Several example results are shown below, all off which return the same result in SQL:

- Zach & Zac – 4
- Lake & Bake – 3
- Brad & Lad – 2
- Horrible & Great – 1

The C# implementation of these rules is shown in **Figure B**.

Figure B



```
public static int Difference(string data1, string data2)
{
    int result = 0;
    string soundex1 = Soundex(data1);
    string soundex2 = Soundex(data2);

    if (soundex1 == soundex2)
        result = 4;
    else
    {
        string sub1 = soundex1.Substring(1, 3);
        string sub2 = soundex1.Substring(2, 2);
        string sub3 = soundex1.Substring(1, 2);
        string sub4 = soundex1.Substring(1, 1);
        string sub5 = soundex1.Substring(2, 1);
        string sub6 = soundex1.Substring(3, 1);

        if (soundex2.IndexOf(sub1) > -1)
            result = 3;
        else if (soundex2.IndexOf(sub2) > -1)
            result = 2;
        else if (soundex2.IndexOf(sub3) > -1)
            result = 2;
        else
        {
            if (soundex2.IndexOf(sub4) > -1)
                result++;

            if (soundex2.IndexOf(sub5) > -1)
                result++;

            if (soundex2.IndexOf(sub6) > -1)
                result++;
        }

        if (soundex1.Substring(0, 1) ==
            soundex2.Substring(0, 1))
            result++;
    }

    return (result == 0) ? 1 : result;
}
```

The Difference Function

One interesting side effect of this rule set is that depending on which word is sent in first you will get different results.

Take, for example, "Brad" and "Zach". If you feed these into the Difference function as "Difference("Brad", "Zach")" you will get the rank of 1. If you feed these into the Difference function as "Difference("Zach", "Brad")" you will get the rank of 2. This is due to searching the second encoding for characters ranges found in the first, and not searching the first encoding for ranges found in the second. This is the same behavior found in the SQL Difference function so I have not corrected it.

Performance

Obviously the Soundex and Difference functions shown above are not lightweight. However, the C# implementation of Soundex shown above can do about 100,000 encodings of average length names in as little as half a second. The same set of encodings done in SQL takes a little over 1 second to complete.

I have no doubt that the performance for the C# implementation could be increased though, and if you figure out how to do that then leave a comment to let the rest of us know how you did it!

The TechRepublic Download

I would suggest that you take advantage of the TechRepublic Download that is associated with this blog post. The download includes a PDF version as well as a Visual Studio project file that contains all of the code for the functionality discussed.

Additional resources

- TechRepublic's Downloads RSS Feed XML
- Sign up for TechRepublic's Downloads Weekly Update newsletter
- Check out all of TechRepublic's free newsletters
- Catch up with all the How do I articles on TechRepublic.

Version history

Version: 1.0

Published: April 18, 2008

Tell us what you think

TechRepublic downloads are designed to help you get your job done as painlessly and effectively as possible. Because we're continually looking for ways to improve the usefulness of these tools, we need your feedback. Please take a minute to drop us a line and tell us how well this download worked for you and offer your suggestions for improvement.

Thanks!

—The TechRepublic Downloads Team