Counting Letters in an Unicode String

தி.ந.ச.வெங்கட ரங்கன்
T.N.C.Venkata Rangan
Chairman & Managing Director
Vishwak Solutions Pvt. Ltd., India

Unicode is now being used worldwide for all types of applications involving multi-languages. It has found wide adoption for sharing documents like Web Pages and other forms of data seamlessly. In all these applications, it is very important we have utility functions to do all the string handling tasks which were commonly available in the ANSI/8-bit world.

One of the problems I faced was when I was doing Tamil Unicode Web Pages. When compared with English, Tamil has longer words used frequently; and also Tamil Text have to be displayed with slightly bigger font-size than nearby English text for easy reading. For getting the design you desire in Web pages, we generally format texts to fit in with our layouts and we use Style Sheets (CSS) to specify exact dimensions. In one of the web layouts I was working, we had Tamil Texts that has to be shown in the left-hand navigation. Navigation panels typically in a web-page are narrow, with say width in the range of 100 pixels or so. The problem was when the words were long than what can normally be fit into the given space, browsers tend to expand the layout to accommodate the given text. Specifying word-wraps don't help here as a single word itself is long. To get around this, we have to programmatically count the number of letters in a word and insert line breaks after 'n' number of letters.

The problem here is, if we use the string length functions included in major programming platforms we get only number of characters based on storage sizes. They don't understand the language and so don't return letter (**எழுத்து**) count, instead they return only count based on character storage. For example if the text is Tamil '**வி**' or '**கொ**', or Hindi '**सा**',

the returned length is '2'. Obviously this is incorrect, as per language grammar it should be counted as '1' letter. The reason it counts two in Unicode is because to get the letter **வி** in Unicode, two characters are combined "**வ**" and "**ி**", which from a storage perspective is '2' storage slots.

To come up with a reusable solution to this problem, this paper presents with implementations in major programming platforms like Microsoft .NET, JavaScript and PERL. Generic Implementations with full source code for all the 3 platforms are made available in my blog[7] at
www.venkatarangan.com/blog/content/binary/UnicodeLengthSourceCode.zip

The Unicode Technical Standard[1] #18, defines Regular Expression support for string handling of Unicode Strings. Though Microsoft .NET, JavaScript and PERL supports Regular Expressions, in this paper I have shown slightly different approaches to solve the problem. The idea being to showcase different ways of Unicode string handling with these platforms.

Let us now look into each of these implementations in detail.

Microsoft .NET Implementation

The .NET Implementation exposes a class "UnicodeString" which has a method "Length()", this returns the number of Tamil letters present in a string.

This method (Figure 1) uses .NET Framework, System.Char.GetUnicodeCategory()[2] method. This method categorizes a Unicode character into a group identified by one of the UnicodeCategory[3] (Defines the Unicode category of a character) values. The official data mapping[4] Unicode characters with Unicode Category values is defined at Unicode.org.

```
Public Function Length() As Integer

        Dim ModifiersCount As Integer     = 0
        Dim StringLength As Integer       = 0
        Dim CurChar As Char
        Dim CurCategory As String         = ""

        StringLength = InputString.Length
        'counting the characters to be excluded
        '(ா ி ு ெ ே ை ொ ோ ௌ ் ள & ஃ ீ)
        For Each CurChar In InputString
            CurCategory = CurChar.GetUnicodeCategory(CurChar).ToString

            If (CurCategory = NONSPACINGMARK) _
                  Or (CurCategory = SPACINGCOMBININGMARK) _
                  Or (CurCategory = CONTROL) _
                  Or (CurCategory = OTHERNOTASSIGNED) Then
                            ModifiersCount += 1
            End If

        Next

        StringLength -= ModifiersCount

        'Handle double counting of ஸ்ரீ and க்ஷ
        StringLength -= GetNumberofKshaAndSri()

        'Calculating the final length by adding ஃ
        StringLength += GetNumberOfAyudham()

        Return StringLength
End Function
```

Figure 1: Microsoft .NET Implementation (Partial code, for full code refer to download)

In the method, we first we find out the total string length, using the normal String.Length method. Then we use GetUnicodeCategory to find out those characters that are Spacing Mark, Spacing Combining Mark, Control Codes and Other Not Assigned Characters. We subtract this count from the total length. The next two lines are basically exceptions. The first exception is to handle ஸ்ரீ (Sri) and க்ஷ (KSha). Both these letters have no specific slot allocated for them in Unicode, but are formed by combination of two existing Unicode Slots. So when you count them, they are normally counted as 2 letters. So we need to subtract them once. The next exception is for Ayudham (ஃ). This is not counted, so we add the number of Ayudhams.

JavaScript Implementation

In the JavaScript Implementation, we take a different approach to solve the problem. We count the Unicode characters that have to be excluded. The characters we need to exclude from counting, with in the Tamil block turns out to be:

- Various signs
  - 0B82 - TAMIL SIGN ANUSVARA
- Dependent vowel signs:
  - 0BBE to ◌ா TAMIL VOWEL SIGN AA
  - 0BBF ◌ி TAMIL VOWEL SIGN I
  - 0BC0 ◌ீ TAMIL VOWEL SIGN II
  - 0BC1 ◌ு TAMIL VOWEL SIGN U
  - 0BC2 ◌ூ TAMIL VOWEL SIGN UU
  - 0BC3 , 0BC4 , 0BC5 <reserved>
  - 0BC6 ◌ெ TAMIL VOWEL SIGN E
  - 0BC7 ◌ே TAMIL VOWEL SIGN EE
  - 0BC8 ◌ை TAMIL VOWEL SIGN AI
- Two-part dependent vowel signs:
  - 0BCA ◌ொ TAMIL VOWEL SIGN O
  - 0BC6 ◌ெ 0BBE ◌ா
  - 0BCB ◌ோ TAMIL VOWEL SIGN OO
  - 0BC7 ◌ே 0BBE ◌ா
  - 0BCC ◌ௌ TAMIL VOWEL SIGN AU
  - 0BC6 ◌ெ 0BD7 ◌ௗ
- Various signs
  - 0BCD ◌் TAMIL SIGN VIRAMA
  - 0BCE, 0BCF, 0BD0, 0BD1, 0BD2, 0BD3, 0BD4, 0BD5, 0BD6 <reserved>
  - 0BD7 ◌ௗ TAMIL AU LENGTH MARK
- 25CC ◌- Dotted CIRCLE (Mapped in Windows Latha Font)

So the logic followed in the Javascript implementation is to count the total length, subtract the exclusions. As in the Microsoft .NET implementation, we handle ஸ்ரீ (Sri) and க்ஷ (KSha) as exceptions. Figure 2, shows the Javascript implementation.

```
function getUnicodeLength(userInput, chkTamilOnly) {
  //chkTamilOnly - Boolean. If false to count non-tamil characters also.

  var TAMIL ='[\u0B80-\u0BFF]';
  var TAMILEXCLUSION='[\u25CC\u0B82\u0BBE-\u0BD7]';

  //KSHA  \u0B95\u0BCD\u0BB7 First three characters in KSHA sequence
  //SRI   \u0BB8\u0BCD\u0BB0\u0BC0

  var TAMILKSHASRI='\u0B95\u0BCD\u0BB7|\u0BB8\u0BCD\u0BB0\u0BC0';

  var rgxTamilRange = new RegExp(TAMIL,"g");
  var rgxExclusion=new RegExp(TAMILEXCLUSION, "g");
  var rgxKSHASRI = new RegExp(TAMILKSHASRI,"g");

  var matchedExclusion = userInput.match(rgxExclusion);
  var matchedKSHASRI   = userInput.match(rgxKSHASRI);
  var matchedTamil     = userInput.match(rgxTamilRange);

  var actualLength    = 0;
  var chkTamilLength  = 0;
  var excludedLength  = 0;
  var KSHASRILength   = 0;

  if( matchedKSHASRI != null )
      KSHASRILength     =     matchedKSHASRI.length;

  if( matchedExclusion != null )
      excludedLength    =     matchedExclusion.length;

  if( matchedTamil != null )
      chkTamilLength    =     matchedTamil.length;

  if ( chkTamilOnly == 1 )
      actualLength = chkTamilLength - KSHASRILength - excludedLength;
  else
      actualLength  = userInput.length - KSHASRILength - excludedLength;

  return actualLength;
}
```

Figure 2: Javascript Implementation

Only JavaScript versions later than 1.3[5] is fully compliant in handling Unicode Strings. This means this Javascript Implementation will work well with all latest Internet Browsers like IE 5.0+, Mozilla 1.6+ & Safari 1.2+, across platforms.

PERL Implementation

In the PERL Implementation, the technique used is very similar to Microsoft .NET implementation. We use the Unicode Category to find out which characters to exclude in counting. The interesting thing in this implementation is that I have used Regular Expressions to do this, unlike in .NET, where I had looped through each character.

```
sub GetUnicodeStringLength {
my $SourceString = @_[0];
#this is to check whether tamil characters alone should be counted#
my $TamilOnly = @_[1];

#finding the full length of string#
my $SourceLength=length($SourceString);

#variable to count non spacing characters     #
my $CountSpcAndNonSpc = 0;

#take clone of string for replacing the characters to be excluded in counting#
#Mc : Spacing Combining Mark, Mn : Non-Spacing Mark, Cc : Control, Cn : Not Assigned

my $SourceStringForReplace=$SourceString;
if($SourceStringForReplace=~m/[\p{Mc}\p{Mn}\p{Cc}\p{Cn}]+/g)
{
 $CountSpcAndNonSpc =$SourceStringForReplace=~s/[\p{Mc}\p{Mn}\p{Cc}\p{Cn}]+//g;
}

#variable to count ஸ்ரீand க்ஷ#
my $CountSriAndKsha=0;

#take clone of source string to replace ஸ்ரீ and க்ஷ#
my $SourceSpecial=$SourceString;

#replace ஸ்ரீ (\x{0bb8}\x{0bcd}\x{0bb0}\x{0bc0}) and க்ஷ (\x{0b95}\x{0bcd}\x{0bb7}) #
if($SourceSpecial=~m/\x{0b95}\x{0bcd}\x{0bb7}|\x{0bb8}\x{0bcd}\x{0bb0}\x{0bc0}/g)
{
$CountSriKsha=$SourceSpecial=~s/\x{0b95}\x{0bcd}\x{0bb7}|\x{0bb8}\x{0bcd}\x{0bb0}\x{0bc0}//g;
}

#if tamil characters alone should be counted#
my $NonTamilCount=0;
if($TamilOnly==1)
{
 my $SourceSpecial = $SourceString;
 $NonTamilCount=$SourceSpecial =~s/\p{^Tamil}//g;
}

my $ResultCount = $SourceLength -($CountSpcAndNonSpc + $CountSriKsha + $NonTamilCount);

return $ResultCount;
}
```

Figure 3: PERL Implementation

Please note the same technique can be used in .NET as well, and the two implementations use different techniques to achieve the same result only to illustrate variety in problem solving.

While developing this solution in PERL, I noticed that using RegEx to count characters in a Unicode String didn't perform correctly all the time. To circumvent this problem, I have replaced the characters to be excluded and then counted the number of characters replaced.

Beginning with version 5.6, Perl[6] uses logically-wide characters to represent strings internally. This means all Unicode String Handling, including Regular Expressions (REGEX) based functions will perform correctly in PERL versions greater than 5.6.

Glyph Count

In the Microsoft .NET Implementation, I have tried to implement a function that returns the count of Glyphs that will be used to display a Tamil Unicode String. It has to be noted that Glyphs and number of Glyphs used is dependent on the Font used, so this glyph count is only indicative and not conclusive.

One application I see for the Glyph count is to design CSS and Layout accurately. This is possible, as knowing Glyph count, we can pre-determine the number of pixels that will be used to display a Tamil String.

```
Public Function GetGlyphCount() As Integer
   'regex for characters with 0 glyph count
    '◌ 0B82 ◌ி 0BBF ◌ீ 0BC0 ◌ு 0BC1 ◌ூ 0BC2 ◌் 0BCD
    'regex for characters with 2 glyph count
    'ெ 0BCA ே 0BCB ை 0BCC

    Dim rgxCountGlyph As Regex
    Dim CountTwoGlyphs As Integer
    Dim CountZeroGlyph As Integer
    Dim GlyphCount As Integer

    rgxCountGlyph = New Regex("[\u0BCA\u0BCB\u0BCC]")
    GlyphCount = InputString.Length
    CountTwoGlyphs = rgxCountGlyph.Matches(InputString).Count
    CountZeroGlyph = rgxCountGlyph.Matches(InputString, _
"[\u0B82\u0BBF\u0BC0\u0BC1\u0BC2\u0BCD]").Count
    GlyphCount += CountTwoGlyphs - CountZeroGlyph
    rgxCountGlyph = Nothing
    Return GlyphCount
End Function
```

Figure 4: Glyph Count Implementation in Microsoft VB.NET

Conclusion

Using these generic implementations, we can easily count the number of Tamil letters present in a Unicode String. This will help us to manage Tamil inputs and ensure we achieve our desired display layouts as easily as we do English Text.

Foot Note from Author

I would like to thank Mr.Muthu Nedumaran, Chairman, INFITT and Dr.K.Kalyanasundaram, Vice Chairman, INFITT for encouraging me to write this paper. Initially I was only working on a quick-fix for this problem, but Mr.Muthu inspired me to work on this complete, multi-platform solution.

I also would like to express my thanks to my colleagues at Vishwak Solutions Pvt. Ltd., Ms.V.Srimathi and Mr.Sathish Kumar. Both of them helped me in coding the PERL and JavaScript implementations.

References

1. Unicode Technical Standard #18 (http://www.unicode.org/reports/tr18/)
2. Char.GetUnicodeCategory Method (.NET Framework) (http://msdn.microsoft.com/library/en-us/cpref/html/frlrfSystemCharClassGetUnicodeCategoryTopic.asp)
3. UnicodeCategory Enumeration (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemglobalizationunicodecategoryclasstopic.asp)
4. The official data mapping Unicode characters to the General Category value is available at http://www.unicode.org/Public/UNIDATA/UnicodeData.txt
5. Javascript and Unicode (http://www.js-x.com/javascript/core_js15/ident.php#1009568)
6. Unicode in PERL (http://aspn.activestate.com/ASPN/docs/ActivePerl/lib/Pod/perlunicode.html)
7. Venkatarangan's Blog (http://www.venkatarangan.com/blog)