Using Google Translate on Windows Phone

This article shows how to use the Google translate web service in Windows Phone applications.

Introduction

I have been browsing the web to find a free translator service, that I can use. I have finally found a solution and implemented it in Windows Phone 8. (I haven't tested on WP7 devices). It uses Yahoo Query Language to access Google translate service . The limit of using this service is 20000 queries/IP address/hour. No registration is required, as it's a public service. See the google.translate table on http://www.datatables.org.

Note: Google Translate API is available as a paid service if used directly. However, through YQL it is freely available as mentioned above.

Implementation

You'll need to add two files to your project. The main class and an event argument class. Additionally, Json.NET is requied for the class to work. It's a free package.

Translator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using Newtonsoft.Json;
using YourNameSpace;

public delegate void TranslatedString(TranslatedStringEventArgs e);

namespace YourNameSpace
{
    class Translator
    {
        public event TranslatedString TranslatedString;
        public string translatingString;

        // Supporting function to make the URI generation simpler.
        private Uri constructUri(string to, string text)
        {
            string url = @"http://query.yahooapis.com/v1/public/yql?q=" +
Uri.EscapeDataString("select * from google.translate where q=\"" + text + "\" and
target=\"" + to + "\";") +
"&format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys&callback=";
            return new Uri(url, UriKind.Absolute);
        }

        public void TranslateString(string to, string text)
        {
            // getting the translation via YQL, setting up a WebClient for this
            WebClient wc = new WebClient();
            wc.OpenReadCompleted += wc_OpenReadCompleted;
            wc.OpenReadAsync(constructUri(to, text));
            translatingString = text;
        }

        void wc_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
        {
            // getting a new class to return and filling in the inital translate string
            TranslatedStringEventArgs tsea = new TranslatedStringEventArgs();
            tsea.initialString = translatingString;

            // checking if the translation succeeded
            if (e.Error == null)
            {
                // setting the return values
                tsea.Error = false;
                tsea.ErrorMessage = "";

                //helper variables for converting
                string resultString = "";
                byte[] byteArrayForResultString = new byte[e.Result.Length];

                //converting the returned value to string - that's what Json.NET eats
                e.Result.Read(byteArrayForResultString, 0,
Convert.ToInt32(e.Result.Length));
                resultString = UTF8Encoding.UTF8.GetString(byteArrayForResultString, 0,
byteArrayForResultString.Length);

                // try to parse the results
                try
                {
                    // doing the actual work
                    Newtonsoft.Json.Linq.JObject obj =
(Newtonsoft.Json.Linq.JObject)JsonConvert.DeserializeObject(resultString);

                    // Since everything is called "json" in the json array (pretty
straightforward, but not that practical if you ask me),
                    // we have to navigate to our string manually.
                    tsea.translatedString =
(((((((((((((((Newtonsoft.Json.Linq.JContainer)(obj)).First).First).Last).First).First).First).First).First).First).First).First).First).First).First).First).ToString();
                }
                // handle the exceptions if there are
                catch (Exception serializer_exception)
                {
                    tsea.Error = true;
                    tsea.ErrorMessage = "Error in JSON Serializing: " +
serializer_exception.Message + Environment.NewLine + resultString;
```


(C) Copyright Nokia 2014. All rights reserved.

```
                }
            }
            else
            {
                tsea.Error = true;
                tsea.ErrorMessage = "Error in WebClient: " + e.Error.Message;
            }
            TranslatedString(tsea);
        }
    }
}
```

TranslatedStringEventArgs.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace YourNameSpace
{
    public class TranslatedStringEventArgs:EventArgs
    {
        public string initialString { get; set; }
        public string translatedString {get; set; }
        public bool Error { get; set; }

        public string ErrorMessage { get; set; }

    }
}
```

Usage

The usage is very simple. As shown below, instantiate the `Translator` object `t` and pass the language and string to be translated as arguments to `TranslateString`.

```
Translator  t = new Translator();
t.TranslatedString += t_TranstlatedString;
t.TranslateString("de","How are you?");
```

Here is the translated event handler,

```
void t_TranstlatedString(TranslatedStringEventArgs e)
{
 if (e.Error == false)
 {
                MessageBox.Show(e.translatedString);
 }
 else
 {
  MessageBox.Show(e.ErrorMessage, ":(", MessageBoxButton.OK);
 }
}
```

Source code

You can download the source code for the example here: File:TranslatorExample.zip

(C) Copyright Nokia 2014. All rights reserved.