عنوان: **تبدیل بلوک‌های یونیکد در زیرنویس برای نمایش در تلویزیون‌ها و پلیرها**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۱/۰۱ ۱۰:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Regular expressions, Subtitle, Unicode

مقدمه

موقعی که سینمای ناطق کار خود را آغاز کرد، بسیاری از مردم از آن استقبال کردند و بسیاری از سینماگران که این استقبال را دیدند، رفته رفته به سمت سینمای ناطق کشیده شدند. ولی در این بین یک مشکلی ایجاد شده بود؛ اینکه ناشنوایان دیگر مانند قدیم یعنی دوران صامت نمی‌توانستند فیلم‌ها را تماشا کنند، پس نیاز بود این مشکل به نحوی رفع شود. از اینجا بود که ایده‌ی زیرنویس شکل گرفت و این مشکل را رفع نمود. بعدها فیلم‌ها انتقال دهنده‌ی فرهنگ و پیوند دهنده‌ی مردم با فرهنگ‌های مختلف شدند ولی تفاوت در زبان باعث می‌شد که این امر به خوبی صورت نگیرد. به همین علت زیرنویس، وظیفه‌ی دیگری را هم پیدا کرد و آن رساندن پیام فیلم با زبان خود مخاطب بود. امروزه تهیه‌ی زیرنویس‌ها توسط بسیاری از افراد که با زبان انگلیسی (آشنایی با یک زبان میانی برای ترجمه زیرنویس) آشنایی دارند رواج پیدا کرده و روزانه نزدیک به صد زیرنویس یا گاها بیشتر با زبان‌های مختلف بر روی اینترنت قرار می‌گیرند. بزرگترین سایتی که در حال حاضر با شهرت جهانی در این زمینه فعالیت دارد سایت subscene.com است.

آشنایی با انواع زیرنویس‌ها

زیرنویس‌ها فرمت‌های مختلفی دارند مانند srt,sub idx,smi و ... ولی در حال حاضر معروف‌ترین و معتبرترین فرمت در بین همه‌ی فرمت‌ها Subrip با پسوند SRT می‌باشد که قالب متنی به صورت زیر دارد:

```
203
00:16:38,731 --> 00:16:41,325
<i>Happy Christmas, your arse
I pray God it's our last</i>
```

که باعث میشود حجم بسیار کمی در حد چند کیلوبایت داشته باشد.

بررسی مشکل ما با زیرنویس در تلویزیون‌ها

یکی از مشکلاتی که ما در اجرای زیرنویس‌ها بر روی تلویزیون‌ها داریم این است که حروف فارسی را به خوبی نمی‌شناسند و در هنگام نمایش با مشکل مواجه می‌شوند که البته در اکثر مواقع با تبدیل زیرنویس از ANSI به Unicode یا UTF-8 مشکل حل می‌شود. ولی در بعضی مواقع تلویزیون یا پلیرها از پشتیبانی زبان فارسی سرباز می‌زنند و زیرنویس را به شکل زیر نمایش می‌دهند.

سلام = م ا ل س

به این جهت ما از یک برنامه به اسم srttouni استفاده می‌کنیم که با استفاده یک روش جایگزینی و معکوس سازی، مشکل ما را حل می‌کند. ولی باز هم این برنامه مشکلاتی دارد و از آنجا که برنامه نویس این برنامه که واقعا کمال تشکر را از ایشان، دارم مشخص نیست، مجبور شدم به جای گزارش، خودم این مشکلات را حل کنم. *مشکلات این برنامه :*

عدم حذف تگ‌ها ، گاها برنامه نویس‌ها از تگ هایی چون Bold,italic,underline,color استفاده می‌کنند که معدود برنامه‌هایی آن را پشتیبانی کرده و تلویزیون و پلیرها هم که اصلا پشتیبانی نمی‌کنند و باعث میشود که متن روی تلویزیون مثل کد html ظاهر شود

بعضی جملات دوبار روی صفحه ظاهر می‌شوند.

تنها یک فایل را در هر زمان تبدیل می‌کند. مثلا اگر یک سریال چند قسمته داشته باشید، برای هر قسمت باید زیرنویس را انتخاب کرده و تبدیل کنید، در صورتی که میتوان دستور داد تمام زیرنویس‌های داخل دایرکتوری را تبدیل کرد یا چند زیرنویس را برای این منظور انتخاب کرد.

نحوه‌ی خواندن زیرنویس با کدنویسی

با تشکر از دوست عزیز ما در این صفحه می‌توان گفت یک کد تقریبا خوب و جامعی را برای خواندن این قالب داریم. بار دیگر نگاهی به قالب یک دیالوگ در زیرنویس می‌اندازیم و آن را بررسی می‌کنیم:

```
203
00:16:38,731 --> 00:16:41,325
<i>Happy Christmas, your arse
I pray God it's our last</i>
```

اولین خط شامل شماره‌ی خط است که از یک آغاز می‌گردد تا به تعداد دیالوگ‌ها، خط دوم، زمان آغاز و پایان دیالوگ مورد نظر است، موقعی که دیالوگ روی صفحه ظاهر میشود تا موقعی که دیالوگ از روی صفحه محو شود که به ترتیب بر اساس ساعت:دقیقه:ثانیه و میلی ثانیه می‌باشد. خطوط بعدی هم متن دیالوگ است است و بعد از پایان متن دیالوگ یک خط خالی زیر آن قرار می‌گیرد تا نشان دهد این دیالوگ به پایان رسیده است. اگر همین خط خالی حذف گردد برنامه‌هایی چون Media player classic خطهای زیری را جز متن دیالوگ قبلی به حساب می‌آورند و شماره خط و زمان بندی دیالوگ بعدی به عنوان متن روی صفحه ظاهر می‌گردند و بعضی playerها هم قاطی کرده و کلا زیرنویس را نمی‌خوانند یا اون خط رو نشون نمیدن مثل Kmplayer و هر کدام رفتار خاص خودشان را بروز می‌دهند.

کد زیر در کلاس SubRipServices وظیفه‌ی خواندن محتوای فایل srt را بر اساس عبارتی که دادیم دارد:

```
private readonly static Regex regex_srt = new
Regex(@"(?<sequence>\d+)\r\n(?<start>\d{2}\:\d{2}\:\d{2},\d{3}) --\> " +
            @"(?<end>\d{2}\:\d{2}\:\d{2},\d{3})\r\n(?<text>[\s\S]*?)\r\n\r\n", RegexOptions.Compiled);

 public string ToUnicode(string lines)
        {

        string subtitle= regex_srt.Replace(lines,delegate(Match m)
             {
                 string text = m.Groups["text"].Value;
                 //1.remove tags
                 text = CleanScriptTags(text);

                 //2.replace letters
                 PersianReshape reshaper = new PersianReshape();
                 text = reshaper.reshape(text);
                 string[] splitedlines = text.Split(new string[] { Environment.NewLine },
StringSplitOptions.None);
                 text = "";
                 foreach (string line in splitedlines)
                 {
                     //3.reverse tags
                     text += ReverseText(reshaper.reshape(line))+Environment.NewLine ;
                 }
                 return
                     string.Format("{0}\r\n{1} --> {2}\r\n", m.Groups["sequence"],
m.Groups["start"].Value,
                             m.Groups["end"]) + text + Environment.NewLine+Environment.NewLine ;
             }
            );

            return subtitle;
        }
```

در اولین خط ما یک Regular Expersion یا یک عبارت با قاعده تعریف کردیم که در اینجا میتوانید با خصوصیات آن آشنا شوید. ما برای این کلاس یک الگو ایجاد کردیم و بر حسب این الگو، متن یک زیرنویس را خواهد گشت و خطوطی را که با این تعریف جور در می‌آیند و معتبر هستند، برای ما باز می‌گرداند.

عبارتهایی که به صورت <name>? تعریف شده‌اند در واقع یک نامگذاری برای هر قسمت از الگوی ما هستند تا بعدا این امکان برای ما فراهم شود که خطوط برگشتی را تجزیه کنیم که مثلا فقط قسمت متن را دریافت کنیم، یا فقط قسمت زمان شروع یا پایان را دریافت کنیم و ...

متد tounicode یک آرگومان متنی دارد (lines) که شامل محتویات فایل زیرنویس است. متد Replace در شی regex_srt با هر بار پیدا کردن یک متن بر اساس الگو در رشته lines دلیگیتی را فرا می‌خواند که در اولین پارامتر آن که از نوع matchEvaluator است، شامل اطلاعات متنی است که بر اساس الگو، یافت شده است. خروجی آن از نوع string می‌باشد که با متن پیدا شده بر اساس الگو جابجا خواهد کرد و در نهایت بعد از چندین بار اجرا شدن، کل متن‌های تعویض شده، به داخل متغیر subtitle ارسال خواهند شد.

کاری که ما در اینجا می‌کنیم این است که هر دیالوگ داخل زیرنویس را بر اساس الگو، یافته و متن آن را تغییر داده و متن جدید را جایگزین متن قبلی می‌کنیم. اگر زیرنویس ما 800 دیالوگ داشته باشد این دلیگیت 800 مرتبه اجرا خواهد شد.

از آنجا که ما تنها می‌خواهیم متن زیرنویس را تغییر دهیم، در اولین خط فرامین این دلیگیت تعریف شده، متن مورد نظر را بر اساس همان گروه‌هایی که تعریف کرده‌ایم دریافت می‌کنیم و در متغیر text قرار می‌دهیم:

```
m.Groups["text"].Value
```

در مرحله‌ی بعدی ما اولین مشکلمان (حذف تگ‌ها) را با تابعی به اسم CleanScriptTags برطرف میکنیم که کد آن به شرح زیر است:

```
private static readonly Regex regex_tags = new Regex("<.*?>", RegexOptions.Compiled);
 private  string CleanScriptTags(string html)
        {
            return regex_tags.Replace(html, string.Empty);
        }
```

کد بالا از یک regular Expression دیگر جهت پیدا کردن تگ‌ها استفاده می‌کند و به جای آن‌ها عبارت "" را جایگزین می‌کند. این کد قبلا در سایت جاری در این صفحه توضیح داده شده است. خروجی این تابع را مجددا در text قرار می‌دهیم و به مرحله‌ی دوم، یعنی تعویض کاراکترها می‌رویم:

```
PersianReshape reshaper = new PersianReshape();
                text = reshaper.reshape(text);
                string[] splitedlines = text.Split(new string[] { Environment.NewLine },
StringSplitOptions.None);
                text = "";
                foreach (string line in splitedlines)
                {
                    //3.reverse tags
                    text += ReverseText(reshaper.reshape(line))+Environment.NewLine ;
                }
```

برای اینکه دقیقا متوجه شویم قرار است چکاری انجام شود بیاید دو گروه یا بلوک مختلف در یونیکد را بررسی کنیم. هر بلوک کد در یونیکد شامل محدوده‌ای از کد پوینت هاست که نامی منحصرفرد برای خود دارد و هیچ کدام از کدپوینت‌ها در هر بلوک یا گروه، اشتراکی با بقیه‌ی بلوک‌ها ندارد. سایت codetable از آن دست سایت‌هایی است که اطلاعات خوبی در مورد کدهای یونیکد دارد. در قسمت Unicode Groups دو گروه برای زبان عربی وجود دارند که در جدول این گروه، هر سطر آن یکی از کدها را به صورت دسیمال، هگزا دسیمال و نام و نماد آن، نمایش می‌دهد. ^

^ ,

```
Arabic Presentation Forms-A
```

^

```
,
```

^

```
Arabic Presentation Forms-B
```

بلوک اول طبق گفته‌ی ویکی پدیا دسته‌ی متنوعی از حروف مورد نیاز برای زبان فارسی، اردو، پاکستانی و تعدادی از زبان‌های آسیای مرکزی است.
بلوک دوم شامل نمادها و نشانه‌های زبان عربی است و در حال حاضر برای کد کردن استفاده نمی‌شوند و دلیل حضور آن برای سازگاری با سیستم‌های قدیمی است.
اگر خوب به مشکلی که در بالا برای زیرنویس‌ها اشاره کردیم دقت کنید، گفتیم حروف از هم جدا نشان داده می‌شوند و اگر به بلوک دوم در لینک‌های داده شده نگاه کنید می‌بینید که حروف متصل را داراست. یعنی برای حرف س 4 حرف یا کدپوینت داراست : **ﺳـ** برای کلماتی مثل *سبد* ، **ـﺴـ** برای کلماتی مثل *شانس* ، **ـﺲ** برای کلماتی مثل *بسیار* ، ولی خود س برای کلمات غیر متصل مثل *ناس* ، البته بعضی حروف یک یا دو حالت می‌طلبند مثل **د** ، **ر** که فقط دو حالت **ـﺪ د** ، **ـﺮ ر** را دارند یا مثل آ که یک حالت دارد.
من قبلا یک کلاس به نام lettersTable ایجاد کرده بودم (و دیگر نوشتن آن را ادامه ندادم) که برای هر حرف، یک آیتم در شیءایی از نوع dictionary ساخته بودم و هر کدپوینت بلوک اول را در آن کلید و کد متقابلش را در بلوک دوم، به صورت مقدار ذخیره کرده بودم (گفتیم که هر نماد در بلوک اول، برابر با 4 نماد در بلوک دوم است؛ ولی ما در دیکشنری تنها مقدار اول را ذخیره می‌کنیم. زیرا کد بقیه نمادها دقیقا پشت سر یکدیگر قرار گرفته‌اند که می‌توان با یک جمع ساده از عدد 0 تا 3، به مقدار هر

کدام از نمادها رسید. البته ناگفته نماند بعضی نمادها 2 عدد بودند که این هم باید بررسی شود). برای همین هر کاراکتر را با کاراکتر قبل و بعد می‌گرفتم و بررسی می‌کردم و از یک جدول دیکشنری دیگر هم به اسم specialchars هم استفاده کردم تا آن کاراکترهایی که تنها دو نماد یا یک نماد را دارند، بررسی کنم و این کاراکترها همان کاراکترهایی بودند که اگر قبل یک حرف هم بیایند، حرف بعدی به آن‌ها نمی‌چسبد. برای درک بهتر، این عبارت مثال زیر را برای حرف س در نظر بگیرید:

مستطیل = چون بین هر دو طرف س حر وجود دارد قطعا باید شکل س به صورت ـسـ انتخاب شود ، حالا مثال زیر را در نظر بگیرید:

دست = دـست که اشتباه است و باید باشد دست یعنی شکل سـ باید صدا زده شود، پس این مورد هم باید لحاظ شود.

نمونه‌ای از کد این کلاس:

```
Dictionary<int ,int>  letters=new Dictionary<int, int>();

   //0=0x0 ,1=1x0 ,2=0x1 ,3=1x1
        private void FillPrimaryTable()
        {
            //آ
            letters.Add(1570, 65153);
            //ا
            letters.Add(1575, 65166);
            //أ
            letters.Add(1571, 65155);
            //ب
            letters.Add(1576, 65167);
            //ت
            letters.Add(1578, 65173);
            //ث
            letters.Add(1579, 65177);
            //ج
            letters.Add(1580, 65181);
.....
}

Dictionary<int,byte> specialchars=new Dictionary<int, byte>();

  private void SetSpecialChars()
        {
            //آ
            specialchars.Add(1570, 0);
            //ا
            specialchars.Add(1575, 0);
            //د2
            specialchars.Add(1583, 1);
            //ذ2
            specialchars.Add(1584, 1);
            //ر2
            specialchars.Add(1585, 1);
            //ز2
            specialchars.Add(1586, 1);
            //ژ
            specialchars.Add(1688, 1);
            //و2
            specialchars.Add(1608, 1);
            //أ
            specialchars.Add(1571, 1);

        }
```

کلاس بالا تنها برای ذخیره‌ی کدپوینت‌ها بود، ولی یک کلاس دیگر هم به اسم lettersCrawler نوشته بودم که متد آن وظیفه‌ی تبدیل را به عهده داشت.

در آن متد هر بار یک حرف را انتخاب می‌کرد و حرف قبلی و بعدی آن را ارسال می‌کرد تا تابع CalculateIncrease آن را محاسبه کرده و کاراکتر نهایی را باز گرداند و به متغیر finalText اضافه می‌کرد. ولی در حین نوشتن، زمانی را به یاد آوردم که اندروید به تازگی آمده بود و هنوز در آن زمان از زبان فارسی پشتیبانی نمی‌کرد و حروف برنامه‌هایی که می‌نوشتیم به صورت جدا از هم بود و همین مشکل را داشت که ما این مشکل را با استفاده از یک کلاس جاوا که دوست عزیزی آن را در اینجا به اشتراک گذاشته بود، حل می‌کردیم. پس به این صورت بود که از ادامه‌ی نوشتن کلاس انصراف دادم و از یک کلاس دقیق‌تر و آماده استفاده کردم.

در واقع این کلاس همین کار بالا را با روشی بهتر انجام می‌دهد. همه‌ی نمادها به طور دقیق‌تری کنترل می‌شوند حتی تنوین‌ها و دیگر علائم، همه نمادها با کدهای متناظر در یک آرایه ذخیره شده‌اند که ما در بالا از نوع Dictionary استفاده کرده بودیم.

تنها کاری که نیاز بود، باید این کد به سی شارپ تبدیل میشد و از آنجایی که این دو زبان خیلی شبیه به هم هستند، حدود ده دقیقه‌ای برای ویرایش کد وقت برد که می‌توانید کلاس نهایی را از اینجا دریافت کنید.
پس خط زیر در متد ToUnicode کار تبدیل اصلی را صورت می‌دهد:

```
PersianReshape reshaper = new PersianReshape();
                text = reshaper.reshape(text);
```

بنابراین مرحله‌ی دوم انجام شد. این تبدیل در بسیاری از سیستم‌ها همانند اندروید کافی است؛ ولی ما گفتیم که تلویزیون یا پلیر به غیر از جدا جدا نشان دادن حروف، آن‌ها را معکوس هم نشان می‌دهند. پس باید در مرحله‌ی بعد آن‌ها را معکوس کنیم که اینکار با خط زیر و صدا زدن تابع ReverseText انجام میگیرد

```
//3.reverse tags
                text = ReverseText(text);
```

از آنجا که یک دیالوگ ممکن است چند خطی باشد، این معکوس سازی برای ما دردسر می‌شد و ترتیب خطوط هم معکوس می‌شد. پس ما با استفاده از کد زیر هر یک خط را شکسته و هر کدام را جداگانه معکوس می‌کنیم و سپس به یکدیگر می‌چسبانیم:

```
string[] splitedlines = text.Split(new string[] { Environment.NewLine }, StringSplitOptions.None);
                text = "";
                foreach (string line in splitedlines)
                {
                    //3.reverse tags
                    text += ReverseText(reshaper.reshape(line))+Environment.NewLine ;
                }
```

همه‌ی ما معکوس سازی یک رشته را بلدیم، یکی از روش‌ها این است که رشته را خانه به خانه از آخر به اول با یک for بخوانیم یا اینکه رشته را به آرایه‌ای از کارکاترها، تبدیل کنیم و سپس با Array.Reverse آن را معکوس کرده و خانه به خانه به سمت جلو بخوانیم و خیلی از روش‌های دیگر. ولی این معکوس سازی‌ها برای ما یک عیب هم دارد و این هست که این معکوس سازی روی نمادهایی چون . یا ! و غیره که در ابتدا و انتهای رشته آمده‌اند و حروف انگلیسی، نباید اتفاق بیفتند. پس می‌بینیم که تابع معکوس سازی هم باز باید ویژه‌تر باشد. ابتدا قسمت‌های ابتدا و انتها را جدا کرده و از آن حذف می‌کنیم. سپس رشته را معکوس می‌کنیم. ولی ممکن هست و احتمال دارد که بین حروف فارسی هم حروف انگلیسی یا اعداد به کار رود که آن‌ها هم معکوس می‌شوند. برای همین بعد از معکوس سازی یکبار هم باید آن‌ها را با یک عبارت با قاعده یافته و سپس هر کدام را جداگانه معکوس کرده و سپس مثل روش بالا Replace کنیم و رشته‌های جدا شده را به ابتدا و انتهای آن، سر جای قبلیشان می‌چسبانیم.
این دو تابع برای معکوس کردن عادی یک رشته به کار می‌روند:

```
private string Reverse(string text)
        {
            return Reverse(text,0,text.Length);
        }

        private string Reverse(string text,int start,int end)
        {
            if (end < start)
                return text;
            string reverseText = "";

            for (int i = end-1; i >=start; i--)
            {
                reverseText += text[i];
            }
            return reverseText;
        }
```

ولی این تابع ReverseText جمعی از عملیات معکوس سازی ویژه‌ی ماست؛ مرحله اول، مرحله دریافت و ذخیره‌ی حروف خاص در ابتدای رشته به اسم پیشوند prefix است:

```
private string ReverseText(string text)
        {
```

```
            char[] chararray = text.ToCharArray();
            string reverseText = "";
            bool prefixcomp = false;
            bool postfixcomp = false;
            string prefix = "";
            string postfix = "";

            #region get prefix symbols
            for (int i = 0; i < chararray.Length; i++)
            {
                if (!prefixcomp)
                {
                    char ch =(char) chararray.GetValue(i) ;
                    if (ch< 130)
                    {
                        prefix += chararray.GetValue(i);
                    }
                    else
                    {
                        prefixcomp = true;
                        break;
                    }
                }
            }
            #endregion
}
```

مرحله‌ی دوم هم دریافت و ذخیره‌ی حروف خاص در انتهای رشته به اسم پسوند postfix است که به این تابع اضافه می‌کنیم:

```
#region get postfix symbols
            for (int i = chararray.Length - 1; i >-1 ; i--)
            {
                if (!postfixcomp && prefix.Length!=text.Length)
                {
                    char ch = (char)chararray.GetValue(i);
                    if (ch < 130)
                    {
                        postfix += chararray.GetValue(i);
                    }
                    else
                    {
                        postfixcomp = true;
                        break;
                    }
                }
            }
            #endregion
```

مرحله‌ی سوم عملیات معکوس سازی روی رشته است و سپس با استفاده از یک Regular Expression حروف انگلیسی و اعداد بین حروف فارسی را یافته و یک معکوس سازی هم روی آن‌ها انجام می‌دهیم تا به حالت اولشان برگردند. کل عملیات معکوس سازی در اینجا به پایان می‌رسد:

```
#region reverse text

            reverseText = Reverse(text, prefix.Length, text.Length-postfix.Length);


            reverseText = unTagetdLettersRegex.Replace(reverseText, delegate(Match m)
            {
                return Reverse(m.Value);
            });
            #endregion
```

تعریف عبارت با قاعده‌ی بالا به اسم unTargetedLetters:

```
private static readonly Regex unTagetdLettersRegex = new Regex(@"[A-Za-z0-9]+", RegexOptions.Compiled);
```

آخر سر هم رشته را به‌علاوه پیشوند و پسوند جدا شده بر می‌گردانیم:

```
return prefix+ reverseText+postfix;
```

کد کامل تابع بدین شکل در می‌آید:

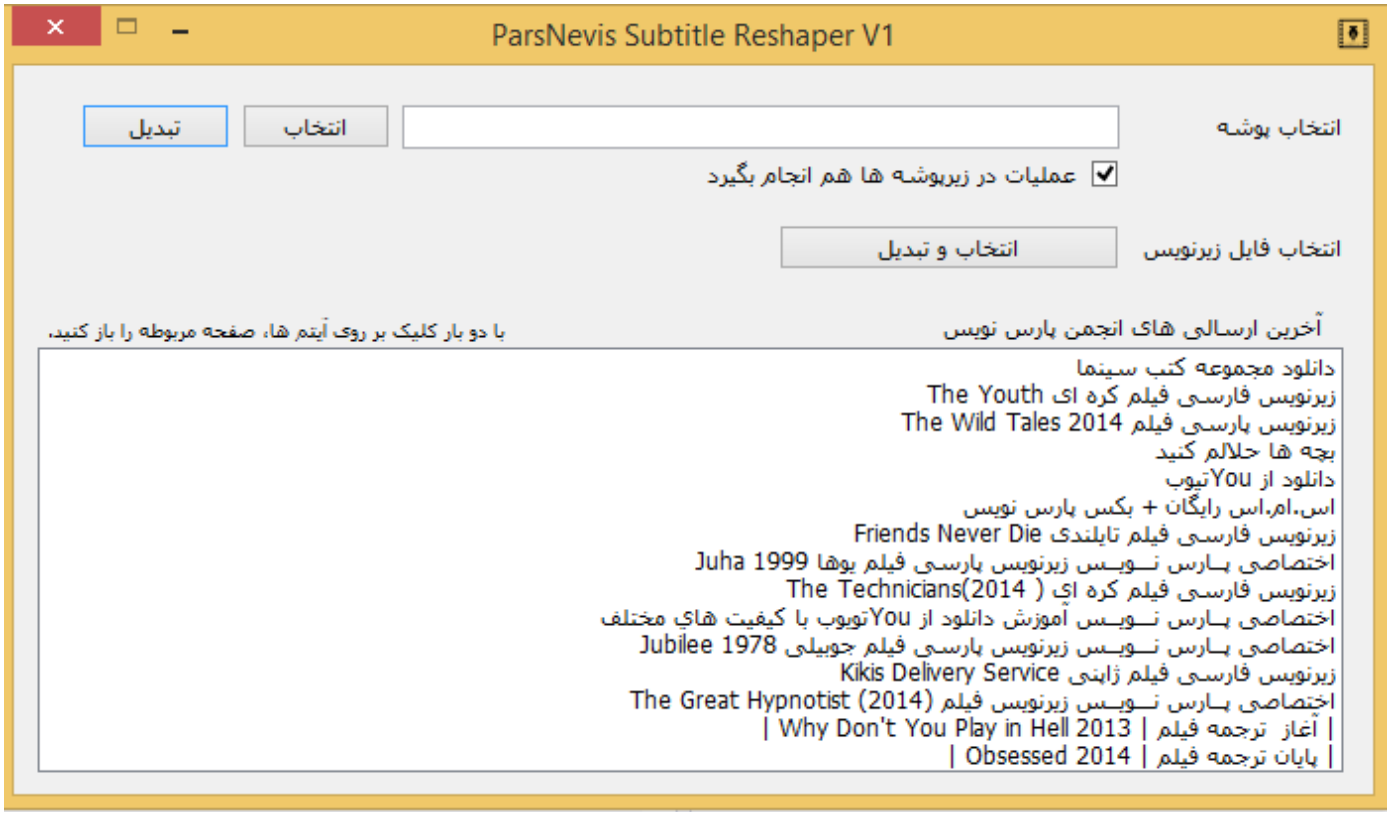
```
private static readonly Regex unTagetdLettersRegex = new Regex(@"[A-Za-z0-9]+", RegexOptions.Compiled);
private string ReverseText(string text)
        {
            char[] chararray = text.ToCharArray();
            string reverseText = "";
            bool prefixcomp = false;
            bool postfixcomp = false;
            string prefix = "";
            string postfix = "";

            #region get prefix symbols
            for (int i = 0; i < chararray.Length; i++)
            {
                if (!prefixcomp)
                {
                    char ch =(char) chararray.GetValue(i) ;
                    if (ch< 130)
                    {
                        prefix += chararray.GetValue(i);
                    }
                    else
                    {
                        prefixcomp = true;
                        break;
                    }
                }
            }
            #endregion

            #region get postfix symbols
            for (int i = chararray.Length - 1; i >-1 ; i--)
            {
                if (!postfixcomp && prefix.Length!=text.Length)
                {
                    char ch = (char)chararray.GetValue(i);
                    if (ch < 130)
                    {
                        postfix += chararray.GetValue(i);
                    }
                    else
                    {
                        postfixcomp = true;
                        break;
                    }
                }
            }
            #endregion

            #region reverse text

            reverseText = Reverse(text, prefix.Length, text.Length-postfix.Length);


            reverseText = unTagetdLettersRegex.Replace(reverseText, delegate(Match m)
            {
                return Reverse(m.Value);
            });
            #endregion



            return prefix+ reverseText+postfix;
        }
```

در نهایت، خط آخر دلیگت همه چیز را طبق فرمت یک دیالوگ srt چینش کرده و بر می‌گردانیم.

```
return
                    string.Format("{0}\r\n{1} --> {2}\r\n", m.Groups["sequence"],
m.Groups["start"].Value,
                        m.Groups["end"]) + text + Environment.NewLine+Environment.NewLine ;
```

رشته subtitle را به صورت srt ذخیره کرده و انکودینگ را هم Unicode انتخاب کنید و تمام.

نمایی از برنامه‌ی نهایی



اجرای زیرنویس تبدیل شده روی کامپیوتر

روی پلیر یا تلویزیون

نکته‌ی نهایی: هنگام تست زیرنویس روی فیلم متوجه شدم پلیر خطوط بلند را که در صفحه‌ی نمایش جا نمی‌شود، می‌شکند و به دو خط تقسیم می‌کند. ولی نکته‌ی خنده دار اینجا بود که خط اول را پایین می‌اندازد و خط دوم را بالا. برای همین این تکه کد را نوشتم و به طور جداگانه در گیت هاب هم قرار داده‌ام.

این تکه کد را هم بعد از

```
//1.remove tags
                text = CleanScriptTags(text);
```

به برنامه اضافه می‌کنیم:

```
text =StringUtils.ConvertToMultiLine(text);
```

از این پس خطوط به طولی بین 30 کاراکتر تا چهل کاراکتر شکسته خواهند شد و مشکل خطوط بلند هم نخواهیم داشت.
کد متد ConvertToMultiline:

```
namespace Utils
{
```

```
    public static class StringUtils
    {
        public static string ConvertToMultiLine(String text, int min = 30, int max = 40)
        {
            if (text.Trim() == "")
                return text;

            string[] words = text.Split(new string[] { " " }, StringSplitOptions.None);

            string text1 = "";
            string text2 = "";
            foreach (string w in words)
            {
                if (text1.Length < min)
                {
                    if (text1.Length == 0)
                    {
                        text1 = w;
                        continue;
                    }

                    if (w.Length + text1.Length <= max)
                        text1 += " " + w;
                }
                else
                    text2 += w + " ";

            }
            text1 = text1.Trim();
            text2 = text2.Trim();
            if (text2.Length > 0)
            {
                text1 += Environment.NewLine + ConvertToMultiLine(text2, min, max);
            }
            return text1;
        }

    }
}
```

آرگومان‌های min و max که به طور پیش فرض 30 و 40 هستند، سعی می‌کنند که هر خط را در نهایت به طور حدودی بین 30 تا 40 کاراکتر نگه دارند.

نکته پایانی : خوشحال میشم دوستان در این پروژه مشارکت داشته باشند و اگر جایی نیاز به اصلاح، بهبود یا ایجاد امکانی جدید دارد کمک حال باشند و سعی کنند تا آنجا که می‌شود برنامه را روی net frame work 2 نگه دارند و بالاتر نبرند. چون استفاده کننده‌های این برنامه کاربران عادی و گاها با دانش پایین هستند و خیلی از آن‌ها هنوز از ویندوز xp استفاده می‌کنند تا در اجرای برنامه خیلی دچار مشکل نشده و راحت برای بسیاری از آن‌ها اجرا شود.

برنامه مورد نظر را به طور کامل می‌توانید از اینجا یا اینجا به صورت فایل نهایی و هم سورس دریافت کنید.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۱ ۱۲:۶

- در فایل‌های PDF هم این چرخاندن حروف برای نمایش صحیح متون فارسی باید انجام شود. در مطلب « استخراج متن از فایل‌های PDF توسط iTextSharp » در انتهای بحث آن، کلاسی بر اساس API ویندوز البته، برای اصلاح این جایگذاری ارائه شده‌است. شاید در این پروژه هم کاربرد داشته باشد. البته در این حالت پروژه تنها در ویندوز قابل اجرا خواهد بود. یا نمونه‌ی دیگر آن فایل bidi.js موزیلا است که در پروژه‌ی PDF آن استفاده شده‌است.
- در یک سری پلیرها به نظر وجود BOM برای خواندن زیرنویس فارسی اجباری است؛ وگرنه فایل را یونیکد تشخیص نمی‌دهند.
- در حین ذخیره سازی از Encoding.Unicode استفاده کرده‌اید (UTF 16 هست در دات نت). شاید Encoding.UTF8 را هم آزمایش کنید، مفید باشد. حجم UTF 16 نسبت به UTF 8 نزدیک به دو برابر است و شاید بعضی پخش کننده‌ها با آن مشکل داشته باشند.
- به روز رسانی نرم افزار و firmware دستگاه هم در بسیاری از اوقات مفید است؛ خصوصا برای رفع مشکلات یونیکد آن‌ها.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۱/۰۱ ۱۵:۸

در مورد انکودینگ طبق گفته شما اون رو به UTF-8 تغییر دادم و دستگاه هم نمایش داد. برنامه رو هم به روز کردم و گستره شکستن جمله رو هم از 40 کاراکتر تا 50 کاراکتر تغییر دادم . چون فکر کنم قبلی جملات رو خیلی کوتاه می‌کرد.
در مورد به روزآوری firmware هم بهتر هست که کاربرها اصلا این کار رو نکنن یا بعد از تحقیق در مورد آپدیت جدید تصمیم بگیرن. چون بسیاری از دستگاه‌ها به خصوص سامسونگ که خودم پلیر BD-d5900 رو دارم بعد از به روز آوری دچار مشکل میشن که این مشکل ویژگی cinavia هست که باعث میشه دستگاه بعضی از فیلم‌ها که شامل این فناوری هستن رو تشخیص بده که کپی هستند. بدین صورت که بعد از 15 تا 20 دقیقه از تماشای فیلم صدا قطع میشه و یک پیام روی صفحه نمایش داده میشه.

به غیر از اون سامسونگ در آپدیت‌ها جدیدش روش‌های مقابله با sammy Go و روت کردن دستگاه رو هم گنجانده که از نصب اون جلوگیری کنه
کلا هیچ خیری در آپدیت این نوع دستگاه وجود نداره، ما هم به امید خواندن بهتر بعضی از کدکها آپدیت کردیم ولی تنها چیزی که گیرمان آمد همین بود و آخرین آپدیتش هم همین بود. حالا یه فکری هم باید برای حل این مشکل کرد حالا با داونگرید یا تغییرکد منطقه.