عنوان: **حذف اعراب از حروف و کلمات**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۱۰ ۰:۱۰
آدرس: www.dotnettips.info
گروه‌ها: Security, Persian, Unicode

برای بهبود قسمت ثبت نام در یک سایت بهتر است بین «وحید» و «وَحید» تفاوتی قائل نشد. این مورد ممکن است خصوصا حین ارسال پیام‌های خصوصی در آینده جهت تشخیص افراد مشکل ساز شود. همچنین در تهیه slug برای نمایش در Urlها نیز باید اعراب را حذف کرد. منظور از slug، عنوان کوتاهی است که در انتهای یک آدرس ممکن است ذکر شود.

```
http://www.site.com/post/12/slug
```

سؤال: چگونه می‌توان اعراب را از متون فارسی یا عربی حذف کرد؟

متد انجام اینکار را در ذیل مشاهده می‌کنید:

```
using System.Globalization;
using System.Text;

    static string RemoveDiacritics(string text)
    {
        var normalizedString = text.Normalize(NormalizationForm.FormD);
        var stringBuilder = new StringBuilder();

        foreach (var c in normalizedString)
        {
            var unicodeCategory = CharUnicodeInfo.GetUnicodeCategory(c);
            if (unicodeCategory != UnicodeCategory.NonSpacingMark)
            {
                stringBuilder.Append(c);
            }
        }

        return stringBuilder.ToString().Normalize(NormalizationForm.FormC);
    }
```

توضیحات

متد Normalize با پارامتر NormalizationForm.FormD، سبب می‌شود تا کاراکترها به گلیف‌های اصلی تشکیل دهنده‌ی آن‌ها تجزیه شوند. به عبارتی، حروف از اعراب جدا خواهند شد. در ادامه این کاراکترها اسکن شده و صرفا مواردی که حروف پایه را تشکیل می‌دهند، جمع آوری و بازگشت داده می‌شوند. حالت NormalizationForm.FormC که در انتها بکار گرفته شده، برعکس است.
در یونیکد یک حرف می‌تواند از یک یا چند code point تشکیل شود. در حالت FormC، هر حرف با اعراب آن یک code point را تشکیل می‌دهند. در حالت FormD، حرف با اعراب آن دو code point را تشکیل خواهند داد. به همین جهت در ابتدای کار، رشته تبدیل به حالت D شده تا بتوان اعراب آن‌را مجزای از حروف پایه حذف کرد.
البته اعراب در اینجا به اعراب عربی ختم نمی‌شود. یک سری حروف اروپایی مانند "ö"، "ä" و "ü" را نیز شامل می‌شود.

نظرات خوانندگان

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۰۲/۱۱ ۱۶:۱۲

اطلاعات بیشتر در این پرسش و پاسخ .
لیست کاراکترهای یونیکد از نوع NonSpacingMark

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۰۲/۱۱ ۱۶:۴۴

یک سوال: علت استفاده از حالت FormC در انتهای کد چیست؟ چرا فقط به کد زیر بسنده نکردیم:

```
return stringBuilder.ToString();
```

بوسیله Normalize، می‌توانیم خروجی را با مقدار string دیگر مقایسه نماییم یا بعبارت دیگر خروجی مقایسه پذیر خواهد شد. در این پرسش و پاسخ بیشتر درباره Normalize بحث شده است.

نویسنده: داوود
تاریخ: ۱۳۹۳/۰۲/۱۳ ۸:۱۳

با سلام
آیا تنوین و تشدید در این حالت جز اعراب محسوب میشوند
و همچنین ی (یای عربی) جز حروف اعراب دار است
تشکر بابت مطلب مفیدتون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۱۳ ۹:۲

- بله.
- خیر.

نویسنده: علیرضا
تاریخ: ۱۳۹۳/۰۲/۱۳ ۱۴:۳۹

با سلام. برای سرچ یک کلمه بدون اعراب در متنی پر از اعراب باید به چه صورت عمل کرد که بهینه باشد؟
مثلا کلمه‌ی محمد را بخواهیم در دیتابیسی که متن کل قرآن است سرچ کنیم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۱۳ ۱۴:۵۶

جستجوی بهینه‌ی متنی بر روی حجم بالایی از اطلاعات بهتر است توسط روش‌های full text search انجام شود. مثلا از لوسین استفاده کنید، به همراه Lucene.Net.Analysis.Analyzer.ArabicAnalyzer آن که مخصوص جستجو بر روی متون عربی است.
همچنین اگر از FTS در SQL Server استفاده می‌کنید باید از accent insensitive collate استفاده کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۲۴ ۲۳:۱۹

اصلاحیه!

کدهای فوق «آ» را تبدیل به «ا» می‌کنند. مشکلی بود که در حین ثبت نام پیش آمده بود. «آفتاب» برای مثال تبدیل به «افتاب»

می‌شد. برای رفع، داخل حلقه:

```
if (unicodeCategory != UnicodeCategory.NonSpacingMark)
                {
                    stringBuilder.Append(c);
                }
                else
                {
                    //اسامی مانند آفتاب نباید خراب شوند
                    if (c == 1619) //آ
                    {
                        stringBuilder.Append(c);
                    }
                }
```

نویسنده: محمد رضا صفری
تاریخ: ۱۳۹۳/۱۲/۰۹ ۱۲:۳۱

با سلام .
من از PersianAnalyzer که قبلا هم معرفی شده بود استفاده می‌کنم .
اما متون من به این صورت هست که حروف فارسی و عربی با هم مخلوط هستند . مثلا یک توضیحی در مورد یکی از آیات قرآن رو فرض کنید .
در حال حاظر مشکلی برای حروف معمولی نیست ، اما حروفی که اعراب دارند پیدا نمی‌شوند . آیا باید از این ArabicAnalyzer به جای اون استفاده کنم یا همون رو میشه جوری انجام داد که مشکلی پیش نیاد ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۲/۰۹ ۱۳:۳

اگر می‌خواهید در حین جستجو، فرقی بین حروف اعراب‌دار و معادل معمولی و بدون اعراب آن‌ها نباشد، در حین تشکیل ایندکس لوسین متد RemoveDiacritics معرفی شده در متن را جهت پاکسازی اعراب، پیش از ذخیره سازی آن‌ها در ایندکس‌ها و اسناد لوسین بکار بگیرد. همچنین در حین جستجو هم ورودی کاربر را با همین متد پاکسازی کنید. برای نمونه معمول است که در حین ایندکس کردن اسناد HTML، ابتدا تمام تگ‌های آن‌ها حذف شده و سپس صرفا متن موجود در صفحه به ایندکس‌ها معرفی می‌شوند.

عنوان: **داستانی از Unicode**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۰/۱۱ ۲۲:۰
آدرس: www.dotnettips.info
گروه‌ها: Unicode, utf-8, ASCII

یکی از مباحثی که به نظرم هر دانشجوی رشته کامپیوتر، فناوری اطلاعات و علاقمند به این حوزه باید بداند بحث کاراکترهاست؛ جدا از اینکه همه ما در مورد وجود ascii یا UTF-8 و ... و توضیحات مختصر آن اطلاع داریم ولی عده‌ای از دوستان مثل من هنوز اطلاعات پایه‌ای‌تر و جامع‌تری در این باره نداریم؛ در این مقاله که برداشتی از وب سایت smashing magazine و W3 است به این مبحث می‌پردازیم.

کامپیوترها تنها با اعداد سر و کار دارند نه با حروف؛ پس این بسیار مهم هست که همه کامپیوترها بر روی یک سری اعداد مشخص به عنوان نماینده‌ای از حروف به توافق برسند. این توافق یکسان بین همه کامپیوترها بسیار مهم هست و باید طبق یک استاندارد مشترک استفاده شود تا در همه سیستم‌ها قابل استفاده و انتقال باشد؛ برای همین در سال 1960 اتحادیه استانداردهای آمریکا، یک سیستم رمزگذاری 7 بیتی را ایجاد کرد؛ به نام American Standard Code for Information Interchange یا کد استاندارد سازی شده آمریکایی برای تبادل اطلاعات یا همان ASCII. این هفت بیت به ما اجازه می‌داد تا 128 حرف را کدگذاری کنیم. این مقدار برای حروف کوچک و بزرگ انگلیسی و هم چنین حروف لاتین، همراه با کدگذاری ارقام و یک سری علائم نگارشی و کاراکترهایی از قبیل tab ، space و موارد مشابه و نهایتا کلیدهای کنترلی کافی بود. در سال 1968 این استاندارد توسط رییس جمهور وقت آمریکا لیندون جانسون به رسمیت شناخته شده و همه سیستم‌های کامپیوتری ملزم به رعایت و استفاده از این استاندارد شدند.

برای لیست کردن و دیدن این کدها و نمادهای حرفیشان می‌توان با یک زبان برنامه نویسی یا اسکریپتی آن‌ها را لیست کرد. کد زیر نمونه‌ای از کد نوشته شده در جاوااسکریپت است.

```
<html>

<body>
 <style type="text/css">p {float: left; padding: 0 15px; margin: 0;}</style>

<script type="text/javascript">
 for (var i=0; i<128; i++) document.writeln ((i%32?'':'<p>') + i + ': ' + String.fromCharCode (i) +
'<br>');

</script>
</body>
 </html>
```

در سال‌های بعدی، با قوی‌تر شدن پردازش‌گرها و 8 بیت شدن یک بایت به جای ذخیره 128 عدد توانستند 256 عدد را ذخیره کنند ولی استاندارد اسکی تا 128 کد ایجاد شده بود و مابقی را به عنوان ذخیره نگاه داشتند. در ابتدا کامپیوترهای IBM از آن‌ها برای ایجاد نمادهای اضافه‌تر و همچنین اشکال استفاده می‌کرد؛ مثلا کد 200 شکل ╚ بود که احتمالا برنامه نویسان زمان داس، این شکل را به خوبی به خاطر میاورند یا مثلا حروف یونانی را اضافه کردند که با کد 224 شکل آلفا α بود و بعدها به عنوان code page 437 نامگذاری شد. هر چند که هرگز مانند اسکی به یک استاندارد تبدیل نشد و بسیاری از کشورها از این فضای اضافی برای استانداردسازی حروف خودشان استفاده می‌کردند و در کشورها کدپیج‌های مختلفی ایجاد شد. برای مثال در روسیه کد پیج 885 از کد 224 برای نمایش Я بهره می‌برد و در کد پیج یونانی 737 برای نمایش حرف کوچک امگا ω استفاده می‌شد. این کار ادامه داشت تا زمانیکه مایکروسافت در سال 1980 کد پیج Windows-1251 الفبای سریلیک را ارئه کرد. این تلاش تا سال 1990 ادامه پیدا کرد و تا آن زمان 15 کدپیج مختلف استانداردسازی شده برای الفبایی چون سیریلیک، عربی، عبری و ... ایجاد شد که این استانداردها از ISO-8859-1 شروع و تا ISO-8859-16 ادامه داشت و موقعی که فرستنده پیامی را ارسال می‌کرد، گیرنده باید از کدپیج مورد نظر مطلع می‌بود تا بتواند پیام را صحیح بخواند.

بیایید با یک برنامه علائم را در این 15 استاندارد بررسی کنیم. تکه کدی که من در اینجا نوشتم یک لیست را که در آن اعداد یک تا 16 لیست شده است، نشان میدهد که با انتخاب هر کدام، کدها را از 0 تا 255 بر اساس هر استاندارد به ترتیب نمایش می‌دهد. این کار توسط تعیین استاندارد در تگ متا رخ میدهد.

در زمان بارگذاری، استانداردها با کد زیر به لیست اضافه می‌شوند.در مرحله بعد لیستی که postback را در آن فعال کرده‌ایم، کد زیر را اجرا می‌کند. در این کد ابتدا charset انتخاب شده ایجاد شده و سپس یکی یکی کدها را به کاراکتر تبدیل می‌کنیم و رشته

نهایی را درج می‌کنیم: (دانلود فایل‌های زیر)

```
private String ISO = "ISO-8859-";
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                for (int i = 1; i < 16; i++)
                {
                    ListItem item = new ListItem();
                    item.Text = ISO + i.ToString();
                    item.Value = i.ToString();
                    DropDownList1.Items.Add(item);
                }
                ShowCodes(1);
            }

        }


        protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (DropDownList1.SelectedItem != null)
            {
                int value = int.Parse(DropDownList1.SelectedValue);
                ShowCodes(value);
            }

        }

        private void ShowCodes(int value)
        {
            Response.Charset = ISO + value;
            string s = "";
            for (int i = 0; i < 256; i++)
            {
                char ch = (char)i;
                s += i + "-" + ch;
                s += "<br/>";//br tag
            }
            Label1.Text = s;
        }
```

تقریبا سال 1990 بود که بسیاری از اسناد به همین شیوه‌ها نوشته و ذخیره شد. ولی باز برای بسیاری از زبان‌ها، حتی داشتن یکی دو حرف بیشتر مشکلاتی را به همراه داشت. مثلا حروف بعضی زبان‌ها مثل چینی و ژاپنی که 256 عدد، پاسخگو نبود و با آمدن شبکه‌ای چون اینترنت و بحث بین المللی شدن و انتقال اطلاعات، این مشکل بزرگتر از آنچه بود، شد.

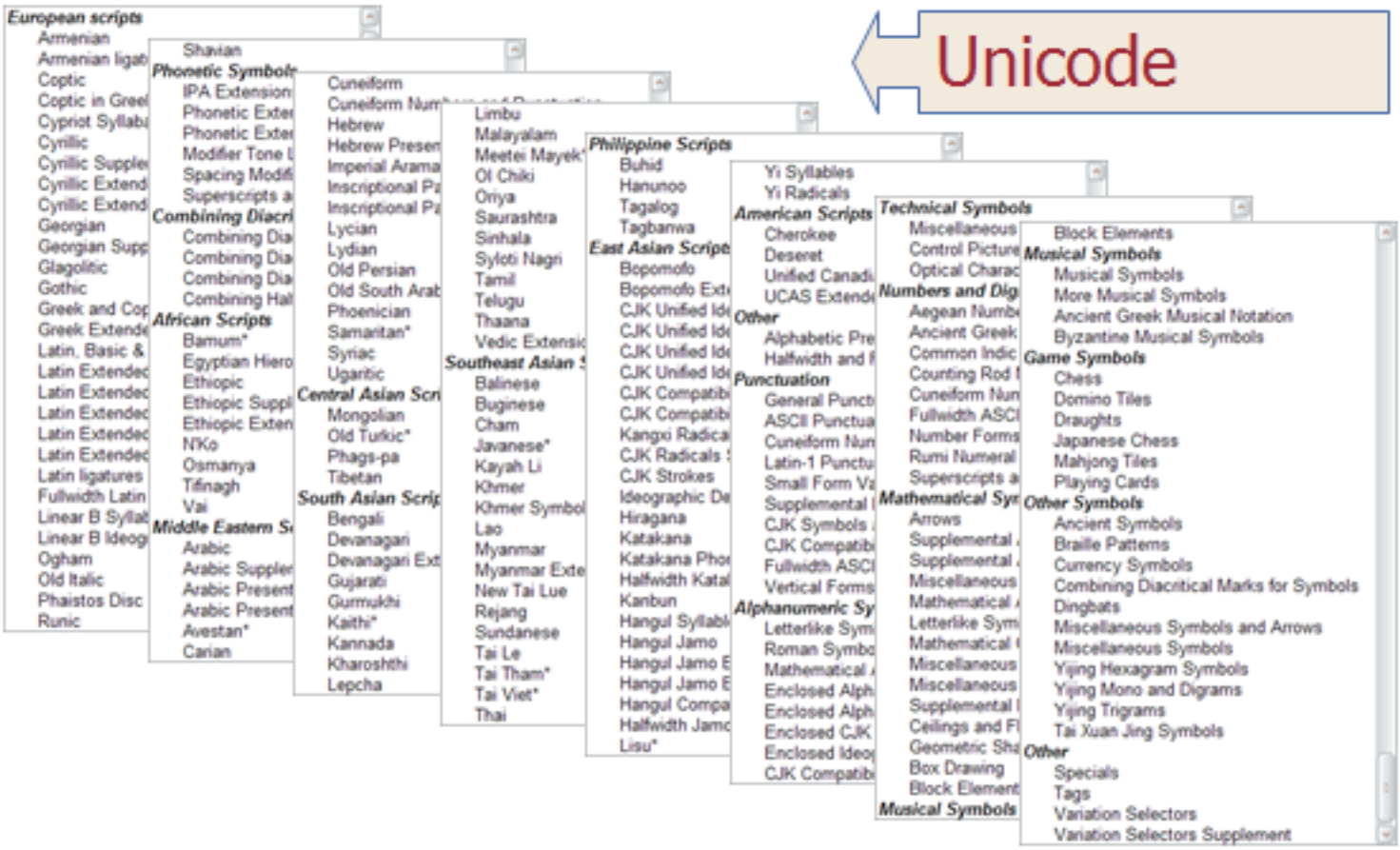
یونیکد نجات بخش

اواخر سال 1980 بود که پیشنهاد یک استاندارد جدید داده شد و در آن به هر حرف و یا نماد در هر زبانی یک عدد یکتا نسبت داده میشد و باید بیشتر از 256 عدد می‌بود که آن را یونیکد نامیدند. در حال حاضر یونیکد نسخه 601 شامل 110 هزار کد می شود. 128 تای آن همانند اسکی است. از 128 تا 255 مربوط به علائم و علامت‌هاست که بیشتر آن‌ها از استاندارد ISO-8859-1 وام گرفته شده‌اند. از 256 به بعد هم بسیاری از علائم تلفظی و ... وجود دارد و از کد 880 زبان یونایی آغاز شده و پس از آن زبان‌های سیریلیک، عبری، عربی و الی آخر ادامه می‌یابند. برای نشان دادن یک کد یونیکد به شکل هگزادسیمال U+0048 نوشته می‌شود و برای تبدیل آن به دسیمال 4*16+8=72 استفاده می‌شود. به هر کد یونیکد، کد پوینت code point گفته میشود.

در ویکی پدیای فارسی، یونیکد اینگونه توضیح داده شده است: *"نقش یونیکد در پردازش متن این است که به جای یک تصویر برای هر نویسه یک کد منحصر به فرد ارایه می‌کند. به عبارت دیگر، یونیکد یک نویسه را به صورت مجازی ارایه می‌کند و کار ساخت تصویر (شامل اندازه، شکل، قلم، یا سبک) نویسه را به عهده نرم‌افزار دیگری مانند مرورگر وب یا واژه‌پرداز می‌گذارد. "*

یونیکد از 8 بیت یا 16 بیت استفاده نمی‌کند و با توجه به اینکه دقیقا 110 ،116 کد را حمایت می‌کند به 21 بیت نیاز دارد. هر چند که کامپیوترها امروزه از معمارهای 32 بیتی و 64 بیتی استفاده می‌کنند، این سوال پیش می‌آید که ما چرا نمی‌توانیم کاراکترها را بر اساس این 32 بیت و 64 بیت قرار بدهیم؟ پاسخ این سوال این‌است که چنین کاری امکان پذیر است و بسیاری از نرم افزارهای نوشته شده در زبان سی و سی ++ از wide character حمایت می‌کنند. این مورد یک کاراکتر 32 بیتی به نام wchar_t است که نوعی داده char توسعه یافته هشت بیتی است و بسیاری از مرورگرهای امروزی از آن بهره مند هستند و تا 4 بیلیون کاراکتر را

حمایت می‌کنند.
شکل زیر دسته بندی از انواع زبان‌های تحت حمایت خود را در نسخه 5.1 یونیکد نشان می‌دهد:



کد زیر در جاوااسکریپت کاراکترهای یونیکد را در مرز معینی که برایش مشخص کرده‌ایم نشان می‌دهد:

```
<html>

<body>
 <style type="text/css">p {float: left; padding: 0 15px; margin: 0;}</style>

<script type="text/javascript">
for (var i=0; i<2096; i++)
   document.writeln ((i%256?'':'<p>') + i + ': ' + String.fromCharCode (i) + '<br>');

</script>
</body>
 </html>
```

CSS & Unicode

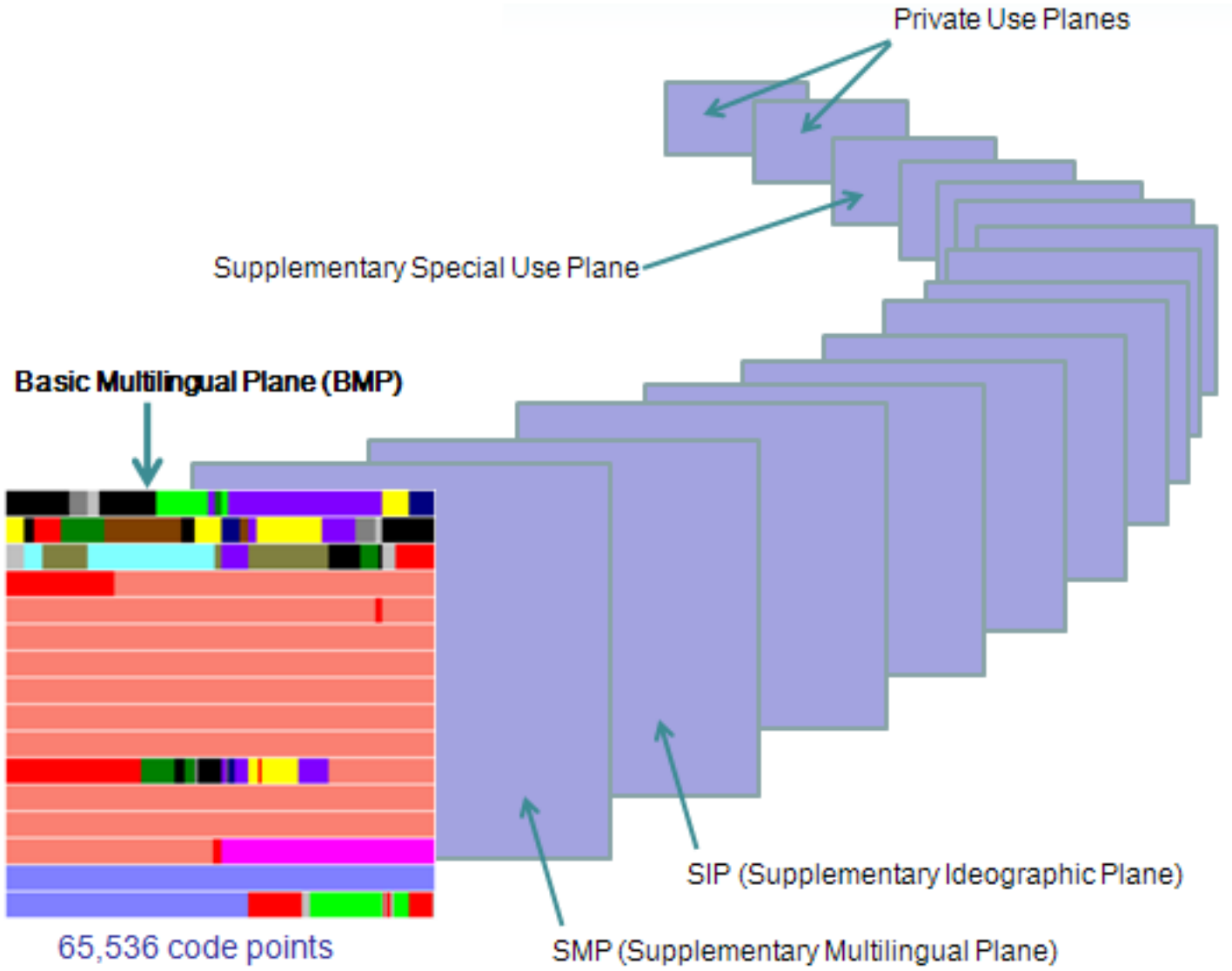
یکی از جذاب‌ترین خصوصیات در css، خصوصیت Unicode-range است. شما میتوانید برای هر کاراکتر یا حتی رنج خاصی از کاراکترها، فونت خاصی را اعمال کنید. به دو نمونه زیر دقت کنید:

```
/* cyrillic */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'),
url(http://fonts.gstatic.com/s/roboto/v14/mErvLBYg_cXG3rLvUsKT_fesZW2xOQ-xsNqO47m55DA.woff2)
format('woff2');
  unicode-range: U+0400-045F, U+0490-0491, U+04B0-04B1, U+2116;
}
```

```
/* greek-ext */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'), url(http://fonts.gstatic.com/s/roboto/v14/-
2n2p-_Y08sg57CNWQfKNvesZW2xOQ-xsNqO47m55DA.woff2) format('woff2');
  unicode-range: U+1F00-1FFF;
}
/* greek */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'),
url(http://fonts.gstatic.com/s/roboto/v14/u0TOpm082MNkS5K0Q4rhqvesZW2xOQ-xsNqO47m55DA.woff2)
format('woff2');
  unicode-range: U+0370-03FF;
}
/* vietnamese */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'),
url(http://fonts.gstatic.com/s/roboto/v14/NdF9MtnOpLzo-noMoG0miPesZW2xOQ-xsNqO47m55DA.woff2)
format('woff2');
  unicode-range: U+0102-0103, U+1EA0-1EF1, U+20AB;
}
/* latin-ext */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'),
url(http://fonts.gstatic.com/s/roboto/v14/Fcx7Wwv8OzT71A3E1XOAjvesZW2xOQ-xsNqO47m55DA.woff2)
format('woff2');
  unicode-range: U+0100-024F, U+1E00-1EFF, U+20A0-20AB, U+20AD-20CF, U+2C60-2C7F, U+A720-A7FF;
}
```

در صورتی که در Unicode-range، تنها یک کد مانند U+20AD نوشته شود، فونت مورد نظر فقط بر روی کاراکتری با همین کد اعمال می‌شود. ولی اگر بین دو کد از علامت - استفاده شود، فونت مورد نظر بر روی کاراکترهایی که بین این رنج هستند اعمال می‌شود U+0025-00FF و حتی می‌توان اینگونه نوشت U+4?? روی کاراکترهایی در رنج U+400 تا U+4FF اعمال می‌شوند. برای اطلاعات بیش‌تر به اینجا و اینجا مراجعه کنید.

به 65536 کد اول یونیکد Basic Multilingual Plan یا به اختصار BMP می‌گویند و شامل همه کاراکترهای رایجی است که مورد استفاده قرار می‌گیرند. همچنین یونیکد شامل یک فضای بسیار بزرگ خالی است که به شما اجازه توسعه دادن آن را تا میلیون‌ها کد می‌دهد. به کاراکترهایی که در این موقعیت قرار می‌گیرند supplementary characters یا کاراکترهای مکمل گویند. برای اطلاعات بیشتر می‌توانید به سایت رسمی یونیکد مراجعه کنید. در اینجا هم مباحث آموزشی خوبی برای یونیکد دارد، هر چند کامل‌تر آن در سایت رسمی برای نسخه‌های مختلف یونیکد وجود دارد.



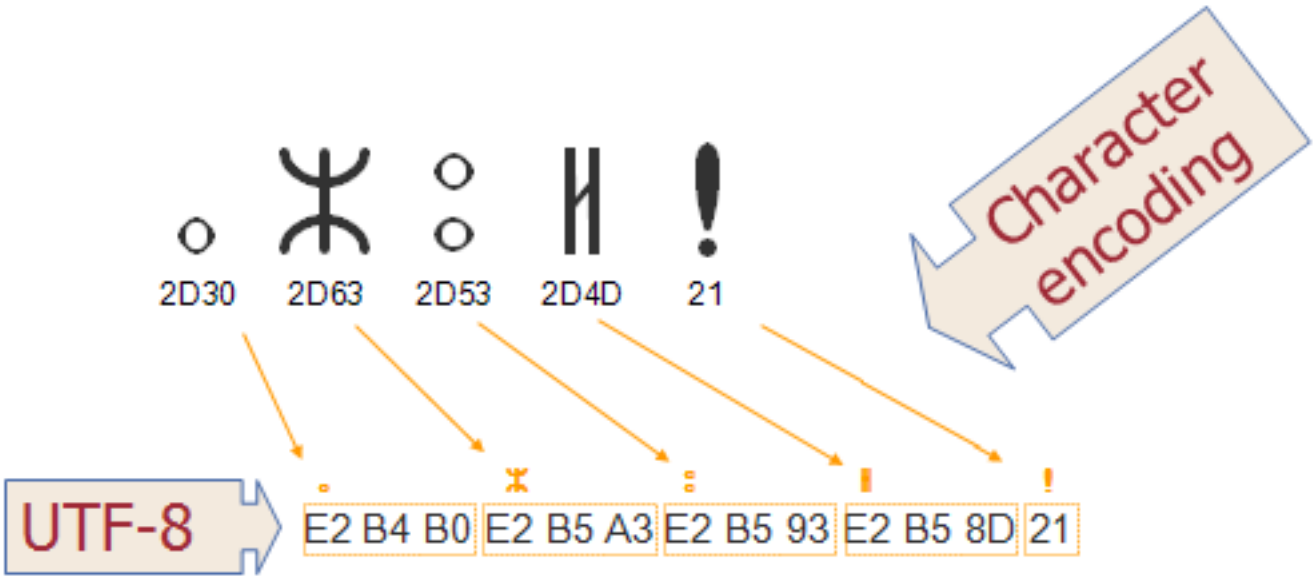
UTF-8 نجات بخش می‌شود

بسیاری از مشکلات ما حل شد. همه حروف را داریم و مرورگرها نیز همه حروف را میشناسند؛ ولی برای ما دو مشکل ایجاد کرده است:

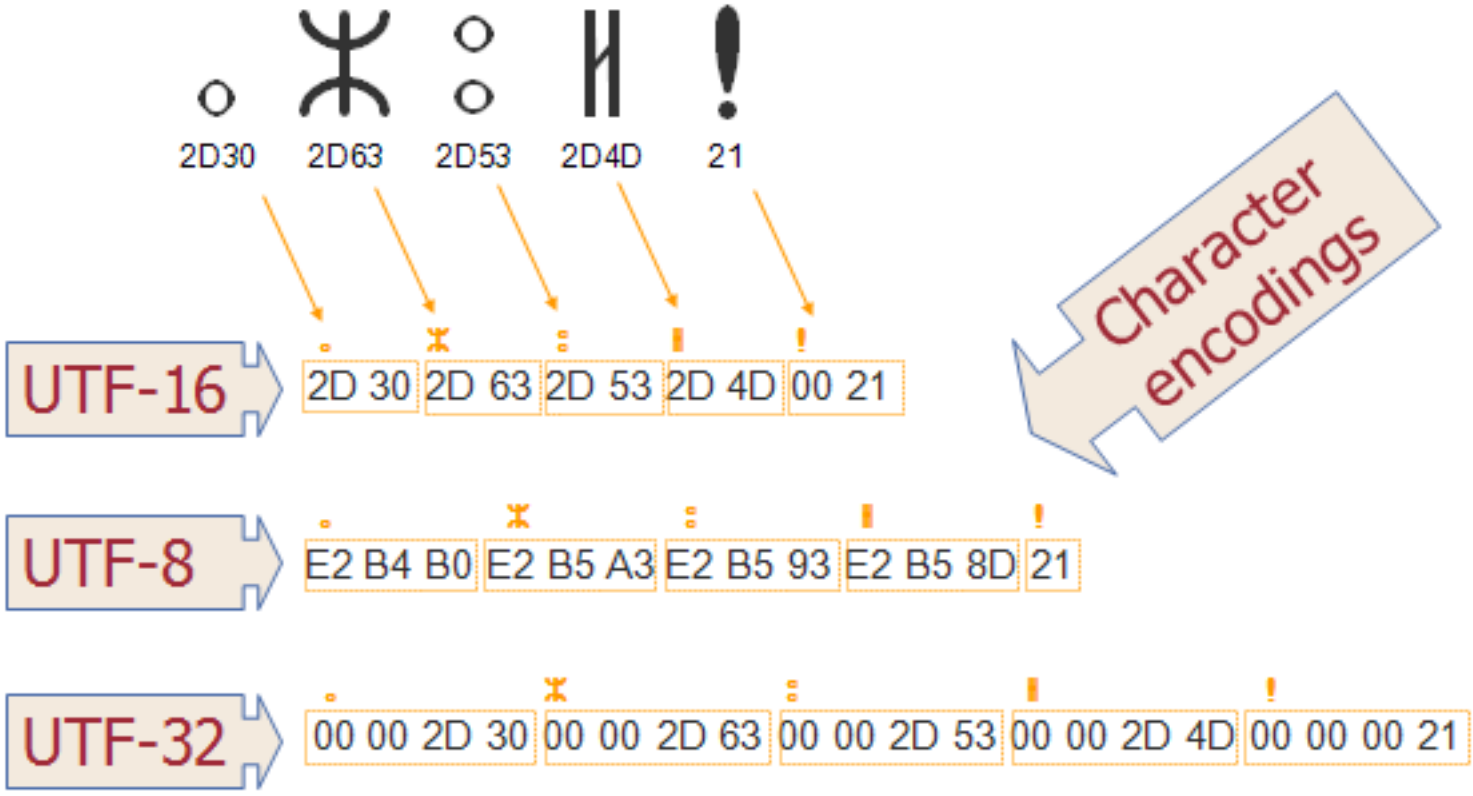
بسیاری از نرم افزارها و پروتکل‌ها هنوز 8 بیتی کار می‌کنند.

اگر یک متن انگلیسی ارسال کنید، 8 بیت هم کافی است ولی در این حالت 32 بیت جابجا می‌شود؛ یعنی 4 برابر و در ارسال و دریافت و پهنای باند برایمان مشکل ایجاد می‌کند.

برای حل این مشکل استانداردهای زیادی چون USC-2 یا UTF-16 ایجاد شدند ولی در سال‌های اخیر برنده رقابت، UTF-8 بود که مخفف عبارت Universal Character Set Transformation Format 8 bit می‌باشد. این کدگذاری بسیار هوشمندانه عمل می‌کند. موقعی که شما کاراکتری را وارد می‌کنید که کدش بین 0 تا 255 است، 8 بیت به آن اختصاص می‌دهد و اگر در محدوده‌ای است که بتوان دو بایت را به آن اختصاص داد، دوبایت و اگر بیشتر بود، سه بایت و اگر باز بیشتر بود 4 بایت به آن اختصاص میدهد. پس با توجه به محدوده کد، تعداد بایت‌ها مشخص می‌شوند. بنابراین یک متن نوشته شده انگلیسی که مثلا از کدهای بین 0تا 128 استفاده می‌کند و فرمت ذخیره آن UTF-8 باشد به ازای هر کارکتر یک بایت ذخیره می‌کند.



مقایسه‌ای بین نسخه‌های مختلف :



همانطور که می‌بینید UTF-8 برای کاراکترهای اسکی، از یک بایت و برای دیگر حروف از دوبایت و برای بقیه BMPها از سه بایت استفاده میکند و در صورتی که کاراکتری در ناحیه مکمل supplementary باشد، از چهار بایت استفاده خواهد کرد. UTF-16 از دو بایت برای نمایش کاراکترهای BMP و از 4 بایت برای نمایش کاراکترهای مکمل استفاده می‌کند و در UTF-32 از 4 بایت برای همه

کاراکترها یا کد پوینت‌ها استفاده می‌شود.

عنوان: **ترفندهای یونیکد برای زبان‌های راست به چپ**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۱۶ ۱۴:۱۰
آدرس: www.dotnettips.info
گروه‌ها: Reporting, Persian, Unicode

آشنایی با RLE

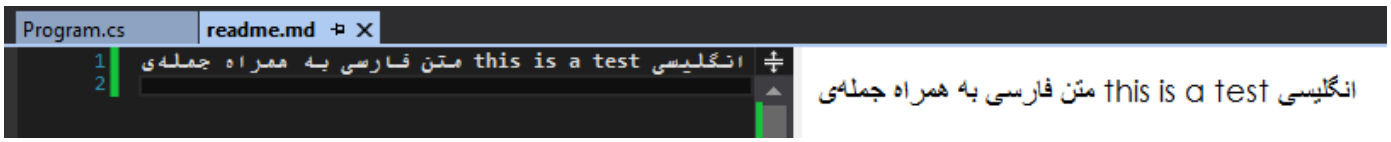
الگوریتم پردازش دوطرفه‌ی یونیکد ، جهت و سمت نمایش متن را بر اساس خواص جهتی هر حرف مشخص می‌کند. در این حالت اگر متن مورد نمایش، انگلیسی و یا فارسی خالص باشند به خوبی عمل می‌کند؛ اما اگر ترکیب این دو را در یک رشته داشته باشیم، نیاز است نحوه‌ی جهت گیری و نمایش حروف را به Unicode bidirectional algorithm معرفی کنیم. این نوع مشکلات را فارسی زبان‌ها در حین نمایش ترکیبی از متن فارسی و انگلیسی در Tooltips، برنامه‌های نمایش زیرنویس‌های فیلم‌ها، برنامه‌های گزارشگیری و امثال آن به وفور مشاهده می‌کنند.
راه حل استاندارد یونیکد آن، استفاده از حروف نامرئی یونیکد است که جهت نمایشی متن جاری را بازنویسی می‌کنند:

```
U+202A:   LEFT-TO-RIGHT EMBEDDING (LRE)
U+202B:   RIGHT-TO-LEFT EMBEDDING (RLE)
U+202D:   LEFT-TO-RIGHT OVERRIDE (LRO)
U+202E:   RIGHT-TO-LEFT OVERRIDE (RLO)
U+202C:   POP DIRECTIONAL FORMATTING (PDF)
```

برای مثال حرف یونیکد نامرئی U202B به این معنا است: «از این لحظه به بعد تا اطلاع ثانوی، متن نمایش داده شده راست به چپ است؛ صرفنظر از خواص جهتی حروف مورد استفاده».
این تا اطلاع ثانوی یا POP نیز توسط حرف U202C مشخص شده و به پایان می‌رسد. به عبارتی یونیکد شبیه به یک پشته یا Stack عمل می‌کند.

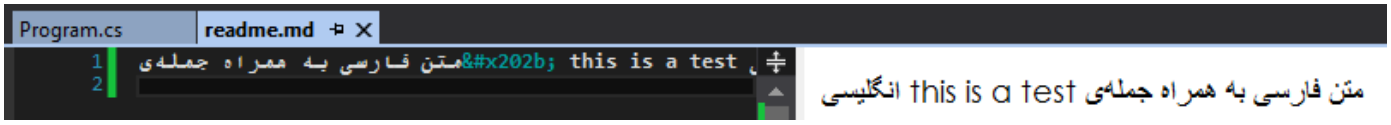
مثال اول

عبارت «متن فارسی به همراه جمله‌ی this is a test انگلیسی» را در نظر بگیرید. اکنون فرض کنید می‌خواهیم از آن جهت ارائه یک فایل readme مخصوص GitHub با فرمت mark down یا md استفاده کنیم:



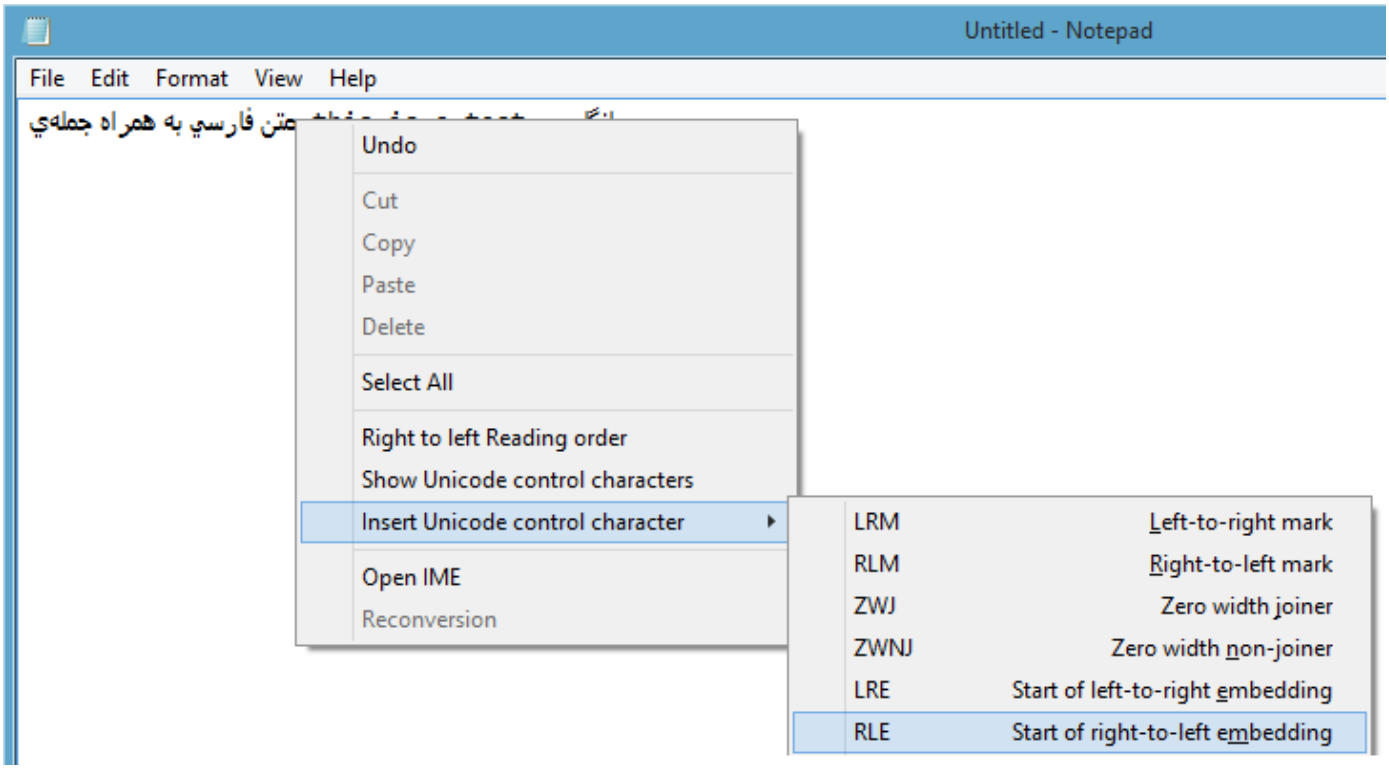
همانطور که ملاحظه می‌کنید، جمله معکوس شده‌است. برای رفع این مشکل می‌توان از کاراکتر نامرئی یونیکد 202b استفاده کرد.
البته در mark down امکان تعریف ساده‌تر این کاراکتر به صورت ذیل نیز پیش بینی شده‌است:

```
&#x202b;
```

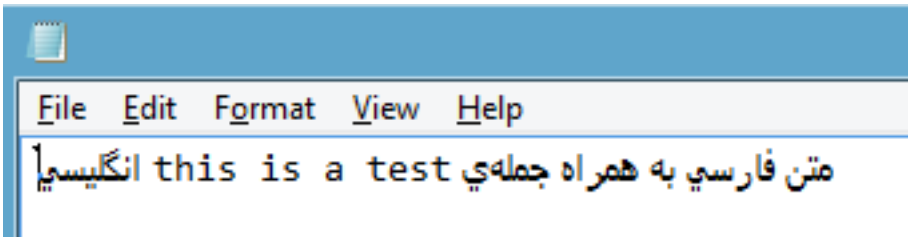


مثال دوم

اغلب نمایشگرهای چپ به راست متون نیز در حالت پیش فرض، عبارت مثال اول را معکوس نمایش می‌دهند:



اگر از notepad استفاده کنید، به صورت توکار امکان افزودن RLE را به ابتدای جمله دارد:



مثال سوم

در زبان‌های دات نتی نیز جهت نمایش صحیح متون ترکیبی، می‌توان حرف RLE را به صورت ذیل به ابتدای یک جمله اضافه کرد:

```
public const char RightToLeftEmbedding = (char)0x202B;
```

این مورد خصوصا در ابزارهای گزارشگیری یا کار با API ویندوز می‌تواند مفید باشد.

تشخیص راست به چپ بودن متن

در محیط وب جهت نمایش صحیح یک متن نیز می‌توان به مرورگرها کمک کرد. تعریف dir=rtl تفاوتی با قرار دادن RLE در ابتدای یک متن ندارد. در این حالت نیاز است بدانیم حروف RTL در چه بازه‌ای از شماره حروف یونیکد قرار می‌گیرند:

```
Right-to-left Unicode blocks for modern scripts are:

 Consecutive range of the main letters:
 U+0590 to U+05FF - Hebrew
 U+0600 to U+06FF - Arabic
 U+0700 to U+074F - Syriac
 U+0750 to U+077F - Arabic Supplement
 U+0780 to U+07BF - Thaana
 U+07C0 to U+07FF - N'Ko
 U+0800 to U+083F - Samaritan

 Arabic Extended:
 U+08A0 to U+08FF - Arabic Extended-A

 Consecutive presentation forms:
 U+FB1D to U+FB4F - Hebrew presentation forms
 U+FB50 to U+FDFF - Arabic presentation forms A

 More Arabic presentation forms:
 U+FE70 to U+FEFF - Arabic presentation forms B
```

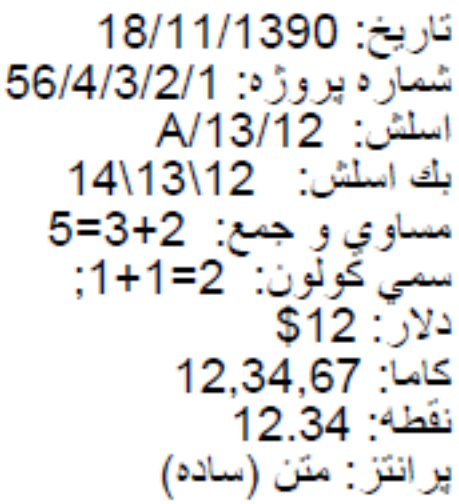
که یک نمونه‌ی ساده شده‌ی این بازه‌ها، به صورت ذیل است:

```
private static readonly Regex _matchArabicHebrew =
new Regex(@"[\u0600-\u06FF,\u0590-\u05FF]", RegexOptions.IgnoreCase | RegexOptions.Compiled);

  public static bool ContainsRtlFarsi(this string txt)
  {
       return !string.IsNullOrEmpty(txt) && _matchArabicHebrew.IsMatch(txt);
  }
```

و حالت پیشرفته‌تر آن‌را که سایت توئیتر برای ارائه‌ی یک جعبه متنی به صورت خودکار راست به چپ شونده، مورد استفاده قرار می‌دهد، در اینجا می‌توانید مطالعه کنید:

RTLText.module.js

نمایش صحیح عبارات ممیز دار در یک گزارش راست به چپ



استاندارد یونیکد یک سری کاراکتر را «کاراکتر ضعیف» معرفی کرده‌است. برای مثال کاراکتر اسلش بکار رفته در یک تاریخ هم از این دست است. بنابراین اگر در یک گزارش تولیدی، شماره کد ممیز دار و یا یک تاریخ را معکوس مشاهده می‌کنید به این علت است که یک «نویسه ضعیف» مثل اسلش نمی‌تواند جهت را تغییر دهد؛ مگر اینکه از یک «نویسه قوی» برای دستکاری آن استفاده شود (مانند RLE و POP که در ابتدای بحث معرفی شدند).
یک مطلب تکمیلی در این مورد: « iTextSharp و نمایش صحیح تاریخ در متنی راست به چپ »
این اصول در تمام محیط‌هایی که از یونیکد پشتیبانی می‌کنند صادق است و تفاوتی نمی‌کند که ویندوز باشد یا Adobe reader و یا یک ابزار گزارشگیری که اصلا برای محیط‌های راست به چپ طراحی نشده‌است.

کار با اعراب در متون راست به چپ

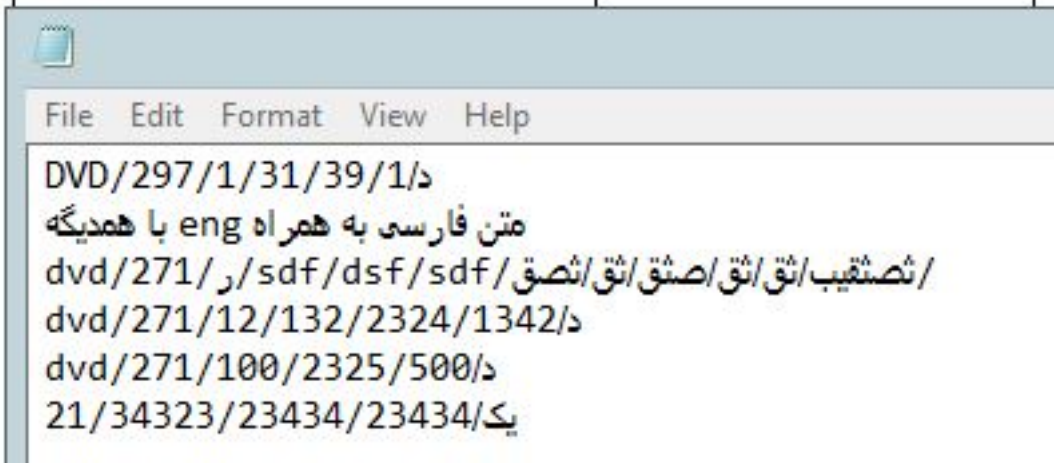
در یونیکد یک حرف می‌تواند از یک یا چند code point تشکیل شود. در حالت FormC، هر حرف، با اعراب آن یک code point را تشکیل می‌دهند. در حالت FormD، حرف با اعراب آن دو code point را تشکیل خواهند داد. به همین جهت نیاز است رشته را تبدیل به حالت D کرد تا بتوان اعراب آن‌را مجزای از حروف پایه، حذف نمود.
البته اعراب در اینجا به اعراب عربی ختم نمی‌شود. یک سری حروف اروپایی مانند "ö"، "ä" و "ü" را نیز شامل می‌شود.
یک مطلب تکمیلی در این مورد: « حذف اعراب از حروف و کلمات »

نظرات خوانندگان

نویسنده: آقا ابراهیم
تاریخ: ۱۳۹۳/۱۰/۱۶ ۱۷:۳۸

سلام. بسیار استفاده بردیم. اما یک سوال؛ من دیتایی مثل تصویر زیر دارم. اما وقتی اعداد وارد میکنم و اسلش میزنم، رشته به هم میریزه:

DVD/297/1/د/1/1/61	د رهی
CD/297/1/د/1/1/62	د رهی
DVD/297/1/د/26/11/4	د رهی
CD/297/2/د/1/1/63	د رهی
CD/297/1/د/32/36/1	د رهی



من میخوام مثل فایل ورد باشه و همه چیز سرجاش. اما وقتی همون فایل ورد کپی می‌گیرم داخل Notpad به هم میزه. از روش شما استفاده کردم. تونستم جمله‌ی فارسی+انگلیسی بنویسم. البته من می‌خوام اول بنویسم dvd/214/CharFarsi/121/452/12. اما همیشه اون بخش CharFarsi میره به آخر. ممنون میشم بهم یاد بدید که چطوری از کاربر بگیرم که به هم نریزه و حتی وقتی سرچ میکنم رشته رو بدون مشکل پیداش کنم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۱۶ ۱۷:۴۹

در مطلب « iTextSharp و نمایش صحیح تاریخ در متنی راست به چپ » متد FixWeakCharacters، برای رفع این مشکل در حین تهیه گزارش‌های PDF ایی، تهیه شد:

```
const char RightToLeftEmbedding = (char)0x202B;
        const char PopDirectionalFormatting = (char)0x202C;

        static string FixWeakCharacters(string data)
```

```
        {
            if (string.IsNullOrWhiteSpace(data)) return string.Empty;
            var weakCharacters = new[] { @"\", "/", "+", "-", "=", ";", "$" };
            foreach (var weakCharacter in weakCharacters)
            {
                data = data.Replace(weakCharacter, RightToLeftEmbedding + weakCharacter +
PopDirectionalFormatting);
            }
            return data;
        }
```

اگر از این متد استفاده نشود، دقیقا خروجی نمایشی PDF اسلش دار، با خروجی نوت پدی که ارائه دادید یکی خواهد بود.
بنابراین همین متد را باید در رخداد on key press و امثال آن، جهت اصلاح جهت ورود کاراکترها فراخوانی کنید. البته این را هم در نظر داشته باشید که برای مثال RLE/POP ایی که در این متد به صورت خودکار درج می‌شود، برای نمایش نهایی طراحی شده‌است (استفاده برای یکبار) و اگر قرار است در on key press فراخوانی شود باید بررسی کنید که آیا قبلا RLE/POP را درج کرده‌اید یا خیر.
همچنین بدیهی است در حین جستجو باید RLE و POP را از رشته‌ی دریافتی حذف کنید (یک Replace ساده با string.Empty)

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۱۰/۳۰ ۱:۳۶

آیا از این روش برای نمایش صحیح Tooltip کامنت راست به چپ کلاس یا متد دات نت در محیط VS می‌توان بهره برد؟

عنوان: **آشنایی با کلیدهای کنترلی و کاربرد آنها**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۲/۲۹ ۲۳:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Unicode

کلیدها یا کاراکترهای کنترلی که در ویکی پدیای فارسی به نویسه‌های کنترلی ترجمه شده اند تنها یک خط تعریف دارند:

یک کاراکتر کنترلی، یک نقطه کدی است که به وسیله علائم نوشتاری قابل نمایش نباشد. مانند **Backspace**

تعریف بالا به ما می‌گوید که در یک متن نوشتاری، به غیر از کد حروفی که مشاهده می‌کنیم، کدهای دیگری هم هستن که قابل نمایش نیستند ولی بین متون وجود دارند. شاید شما تعدادی از آن‌ها را بشناسید مثل کدهای 10 و 13 برای خط بعد و اول سطر که به `line feed` و `carriage return` معروف هستند. در این نوشتار قصد داریم با تعدادی از آنها آشنا شویم.

قبل از آغاز این نوشتار به شما توصیه می‌کنم یک نگاه اجمالی هم که شده بر نوشتار « داستانی از unicode » داشته باشید تا اطلاعات تکمیلی‌تری از این نوشتار به دست آورید. مبحث کلیدهای کنترلی از زمانی آغاز شد که کدهای اسکی ایجاد شدند و به دو دسته‌ی c0 و c1 تقسیم شدند. خود کدهای اسکی هم بر اساس کدهای تلگراف ایجاد شدند و بسیاری از کلیدهای کنترلی هم از آنجا به استاندارد اسکی پیوستند و برای ارتباط و کنترل دستگاه‌هایی چون چاپگرها و تهیه اطلاعات متا در مورد طلاعاتی که قرار بود در نوار مغناطیسی ذخیره شوند به کار رفتند. به عنوان نمونه کد 10 به عنوان `line feed` در چاپگر، یک خط کاغذ را به سمت داخل می‌کشید و کد 13 هم باعث می‌شد چاپگر به ابتدای سطر بازگردد. البته بیشتر این کاراکترها در پردازش متون به خصوص امروزه استفاده نمی‌شوند و فقط یک سری از آن‌ها رایج هستند؛ مثل دو موردی که در بالا و در همین خط به آن‌ها اشاره شد.
دسته‌ی c0 از کد 0 آغاز شده و تا کد 31 ادامه می‌یابد. دو کد بعدی که کدهای Space و DEL هستند در هیچ گروهی قرار نمی‌گیرند.
گروه c1 از کدهای 128 آغاز شده و تا 159 ادامه می‌یابند که جدول این گروه‌ها و کلیدها کنترلی را می‌توانید مشاهده کنید. برای مثال اولین کلید کنترلی که کد آن 0 است به نام نال است که در قدیم هم برای بستن رشته‌ها در زبان سی از آن استفاده می‌کردیم.
هر چند به مرور زمان هم تعدادی از همین کلیدهای کنترلی کاربرد خود را از دست دادند و برای آن‌ها شکلک هایی چون خنده، قلب، نت موسیقی و ... را قرار دادند ولی گاهی اوقات برنامه نویس‌ها هنوز در برنامه‌های خود از کد آن‌ها برای کارهایی چون انجام عملیات بیتی استفاده می‌کنند.

استفاده‌های C0

کلیدهای کنترلی این دسته بیشتر برای منظم ساختن متن‌های ساده و همچنین ایجاد ارتباط در پروتکل ارتباطی و دستگاه‌های مختلف به کار می‌رفت؛ ارسال فرمان‌هایی چون آغاز و توقف کار یا انجام عملی خاص توسط هر یک از این کلیدها صورت می‌گرفت. دستگاه‌هایی چون کارت پانچ‌ها، ماشین تایپ و موارد مشابه، از این نوع هستند. با اینکه عمر این دستگاه به سر آمد ولی کلیدهای کنترلی جان سالم به در بردند.

استفاده‌های C1

این دسته در اواخر سال 1970 اضافه شدند و بیشتر برای ارتباط با چاپگر و صفحه‌ی نمایش به کار می‌رفتند؛ مثل پیمایش‌های افقی و عمودی، تعریف ناحیه‌ای برای پر کردن فرم و Line-Break و کلیدهای انتقالی (شیفت) برای پشتیبانی از کلیدهای کنترلی و قابل چاپ بیشتر. 2 تا از کلیدها هم برای استفاده‌ی خصوصی برنامه نویس کنار گذاشته شدند و 4 تا هم رزرو شده برای استفاده‌ی آینده، تا بعدا استانداردسازی شوند.

کلیدهای کنترلی در سی شارپ

بسیاری از ما از علامت \ در کدهایمان برای قرار دادن کلیدهای کنترلی استفاده می‌کنیم مثل \r\n که ترکیب دو کد CR و LF است.

برای شناسایی یک کلید کنترلی در سی شارپ از متد ایستای Char.IsControl استفاده می‌نماییم. کد زیر در مجموعه‌ی MSDN برای نشان دادن قابلیت این متد نوشته شده است که در طی یک حلقه رنجی از کد پوینت‌ها را بررسی کرده و نتیجه را به صورت شش ستونی در کنسول نمایش می‌دهد. یا کد مشابه دیگر که بر اساس دسیمال نمایش می‌دهد.

```
using System;

public class ControlChars
{
   public static void Main()
   {
      int charsWritten = 0;

      for (int ctr = 0x00; ctr <= 0xFFFF; ctr++)
      {
         char ch = Convert.ToChar(ctr);
         if (char.IsControl(ch))
         {
            Console.Write(@"\U{0:X4}    ", ctr);
            charsWritten++;
            if (charsWritten % 6 == 0)
               Console.WriteLine();
         }
      }
   }
}
// The example displays the following output to the console:
//       \U0000    \U0001    \U0002    \U0003    \U0004    \U0005
//       \U0006    \U0007    \U0008    \U0009    \U000A    \U000B
//       \U000C    \U000D    \U000E    \U000F    \U0010    \U0011
//       \U0012    \U0013    \U0014    \U0015    \U0016    \U0017
//       \U0018    \U0019    \U001A    \U001B    \U001C    \U001D
//       \U001E    \U001F    \U007F    \U0080    \U0081    \U0082
//       \U0083    \U0084    \U0085    \U0086    \U0087    \U0088
//       \U0089    \U008A    \U008B    \U008C    \U008D    \U008E
//       \U008F    \U0090    \U0091    \U0092    \U0093    \U0094
//       \U0095    \U0096    \U0097    \U0098    \U0099    \U009A
//       \U009B    \U009C    \U009D    \U009E    \U009F
```

آیا هنوز برنامه نویس‌ها از کلیدهای کنترلی استفاده می‌کنند؟

این سوال بستگی به برنامه‌ای دارد که شما می‌نویسید. باید گفت هنوز بسیاری از آن‌ها در بسیاری از برنامه‌ها استفاده می‌شوند. مانند بعضی از درایورها برای ارسال اطلاعات به سمت یک قطعه یا دستگاه یا حتی از شما می‌خواهند برنامه‌ای بنویسید که با دستگاه‌های قدیمی ارتباط برقرار کند. برنامه‌هایی که نیاز به کار با رشته‌ها دارند و ...

لیست زیر مشخص می‌کند که کدامیک از کلیدهای کنترلی تا چه اندازه امروزه توسط برنامه نویسان استفاده می‌شوند.

Null	استفاده روزمره‌ای از آن در همه‌ی برنامه‌ها وجود دارد و نیاز به معرفی ندارد.
Transmission Control	SOH , ACK , DLE , ENQ , این کلیدها که 10 عدد هستند شامل EOT , ETB , ETX , Nak , STX , SYN هستند. کاربردشان در انتقال اطلاعات بود ولی امروزه استفاده از آن‌ها به شدت کم شده است و انتقال داده‌ها با سوکت TCP/IP و HTTP و FTP و دیگر پروتکول‌ها به سرانجام رسید و گاها برای بعضی کاربردهای ویژه استفاده می‌شوند.
BEL	این مورد واقعا کاربردش را از دست داده است. وظیفه قبلی‌اش ارسال یه هشدار یا یک زنگ اخطار به کاربر بود. مثلا برای اینکه ماشین تایپ به کاربر هشدار بدهد به آخر خط رسیده است، یک کد BELL به سمت آن ارسال می‌کرد.
Format Effectors	کدهای این دسته عبارتند از BS , CR , FF , HT , **HTJ , HTS , IND** , LF , **NEL , PLD , PLU , RI** , VT , **VTS** هستند که احتمالا مهمترین کدهایی هستند که امروزه از آن‌ها استفاده می‌شود. کاربردشان در فرمت بندی یا قالب بندی متون نوشته شده یا همان کلیدهای قابل چاپ می‌باشد. CR و LF که همیشه معرف حضور ما هستند

Null	استفاده روزمره‌ای از آن در همه‌ی برنامه‌ها وجود دارد و نیاز به معرفی ندارد.
	و بودنشان در سیستم یک امر حیاتی است. HT که همان tab است. BS که همان Backspace است. FF و VT هم که امروزه به ندرت استفاده می‌شوند.
Device Control	هنوز برای ارتباط با دستگاه‌های مختلف مثل کار با پورتها استفاده می‌شوند. کلیدهای معروف آن DC1 و DC3 هستند که به XON و XOFF هم شناخته می‌شوند. یکی از کاربردهای آن.
SUB	یک نماد جایگزین که استفاده‌ی خود را از دست داده است. موقعیکه نمادی نامعتبر بود یا خطایی رخ می‌داد، این نماد جایگزین آن می‌شد. امروزه بیشتر از علامت ؟ در متون استفاده می‌شود. در یک صفحه کلید استاندارد این کد توسط فشرده شدن Ctrl+Z ارسال می‌شود.
CAN , EM	کاربردی امروزه ندارد. CAN برای کنترل خطا به کار می‌رفت و EM در نوارهای مغناطیسی.
Information Separators	شامل 4 کلید RS , GS, FS و US می‌شود که برای جداسازی داده‌ها از یکدیگر به کار می‌روند؛ ولی به‌خاطر جایگزینی آن‌ها با اسنادی مثل XML یا دیتابیس‌ها، استفاده از آن‌ها تا حدودی به پایان رسیده است.
SP	همان کلید space است که نیاز به معرفی ندارد و کارش گویای همه چیز هست.
DELL	همان کلید Delete است.
NBSP	این کلید همان کاراکتر است که در کدهای HTML استفاده می‌شود.
SHY	علامت - یا Hyphen است که به شدت استفاده از آن کم شده است.

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* ، **ﺴـ** برای کلماتی مثل *شانس* ، **ﺴ** 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 **ﺪ د** ، **ﺮ 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 = دـسـت 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:

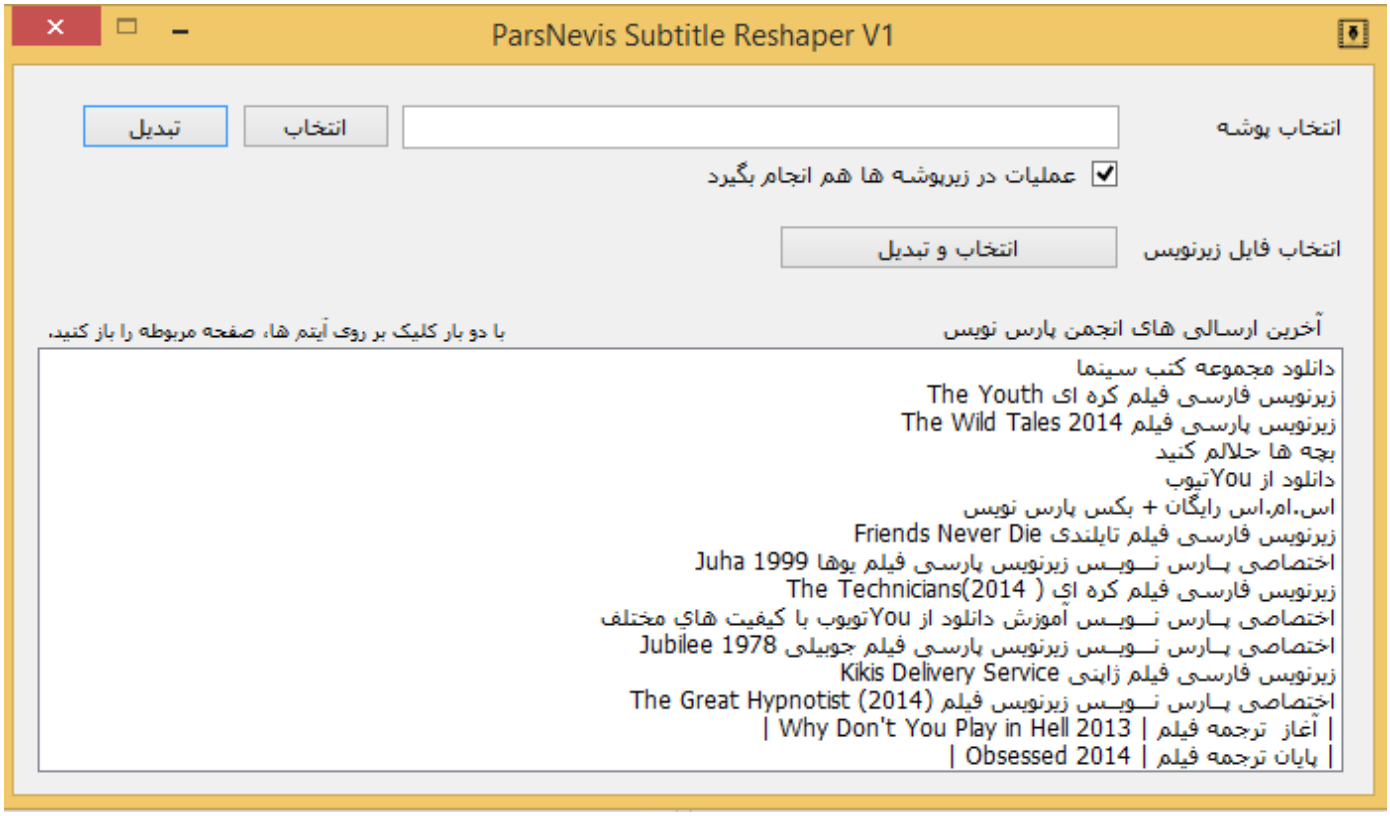
```
private static readonly Regex unTagetdLettersRegex = new Regex(@"[A-Za-z0-9]+", RegexOptions.Compiled);
private string ReverseText(string text)
        {
            char[] chararray = text.ToCharArray();
            string reverseText = "";
            bool prefixcomp = false;
            bool postfixcomp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