

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: ۱۶:۱۲ ۱۳۹۳/۰۲/۱۱

اطلاعات بیشتر در [این پرسش و پاسخ](#) .
[لیست کاراکترهای یونیکد](#) از نوع NonSpacingMark

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

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: ۸:۱۳ ۱۳۹۳/۰۲/۱۳

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: ۹:۰۲ ۱۳۹۳/۰۲/۱۳

- بله.
- خیر.

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

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: ۱۴:۵۶ ۱۳۹۳/۰۲/۱۳

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: ۲۳:۱۹ ۱۳۹۳/۰۵/۲۴

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);
    }
}
```

عنوان:	داستانی از Unicode
نویسنده:	علی یگانه مقدم
تاریخ:	۲۲:۰ ۱۳۹۳/۱۰/۱۱
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](http://smashingmagazine.com) 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 `ASCII` 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 `space`، `tab` و موارد مشابه و نهایتاً 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. 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؛ مثلاً کد 200 شکل ۱ بود که احتمالاً برنامه نویسان زمان داس، این شکل را به خوبی به خاطر می‌آوردند یا مثلاً 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 کدپیچ‌های مختلفی ایجاد شد. برای مثال در روسیه کد پیچ 885 از 224 برای نمایش ۸ بهره می‌برد و در کد پیچ یونانی 737 برای نمایش حرف کوچک امگا ω استفاده می‌شد. این کار ادامه داشت تا زمانی که مایکروسافت در سال 1980 کد پیچ Windows-1251 الفبای سریلیک را ارائه کرد. این تلاش تا سال 1990 ادامه پیدا کرد و تا آن زمان 15 کدپیچ مختلف استانداردسازی شده برای الفبایی چون سریلیک، 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 کدپیچ مورد نظر مطلع می‌بود تا بتواند پیام را صحیح بخواند.

بیباید 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 `meta` 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;
}
```

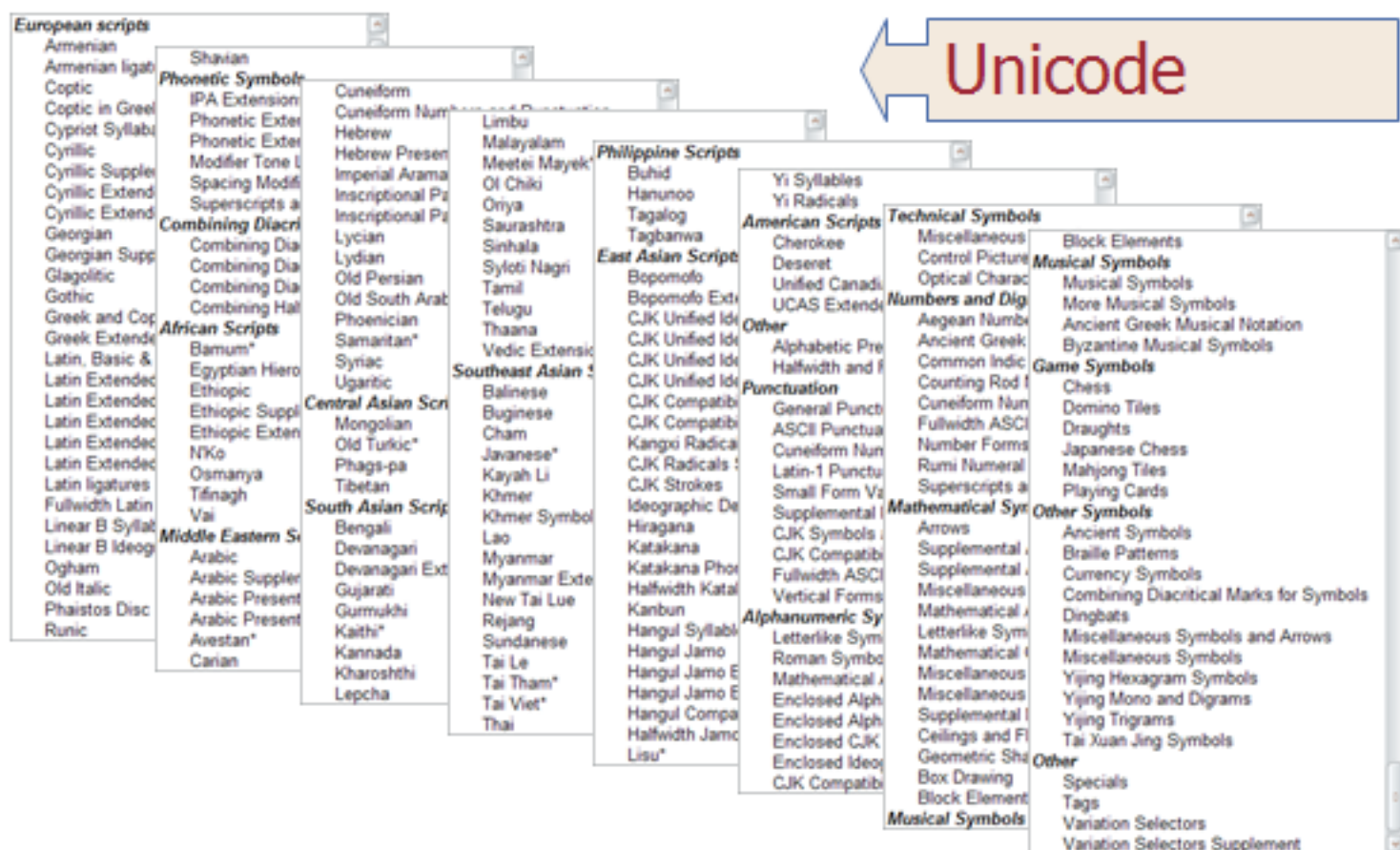
تقریباً سال 1990 بود که بسیاری از اسناد به همین شیوه‌ها نوشته و ذخیره شد. ولی باز برای بسیاری از زبان‌ها، حتی داشتن یکی دو حرف بیشتر مشکلاتی را به همراه داشت. مثلاً حروف بعضی زبان‌ها مثل چینی و ژاپنی که 256 عدد، پاسخگو نبود و با آمدن شبکه‌ای چون اینترنت و بحث بین‌المللی شدن و انتقال اطلاعات، این مشکل 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 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](#) می‌شود. 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 یونانی 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 $72 = 8 + 16 * 4$ 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 دقیقاً 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 این است که چنین کاری امکان‌پذیر است و بسیاری از نرم‌افزارهای 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 بهره‌مند 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_fesZW2x0Q-xsNqO47m55DA.woff2)
  format('woff2');
  unicode-range: U+0400-045F, U+0490-0491, U+04B0-04B1, U+2116;
}
```

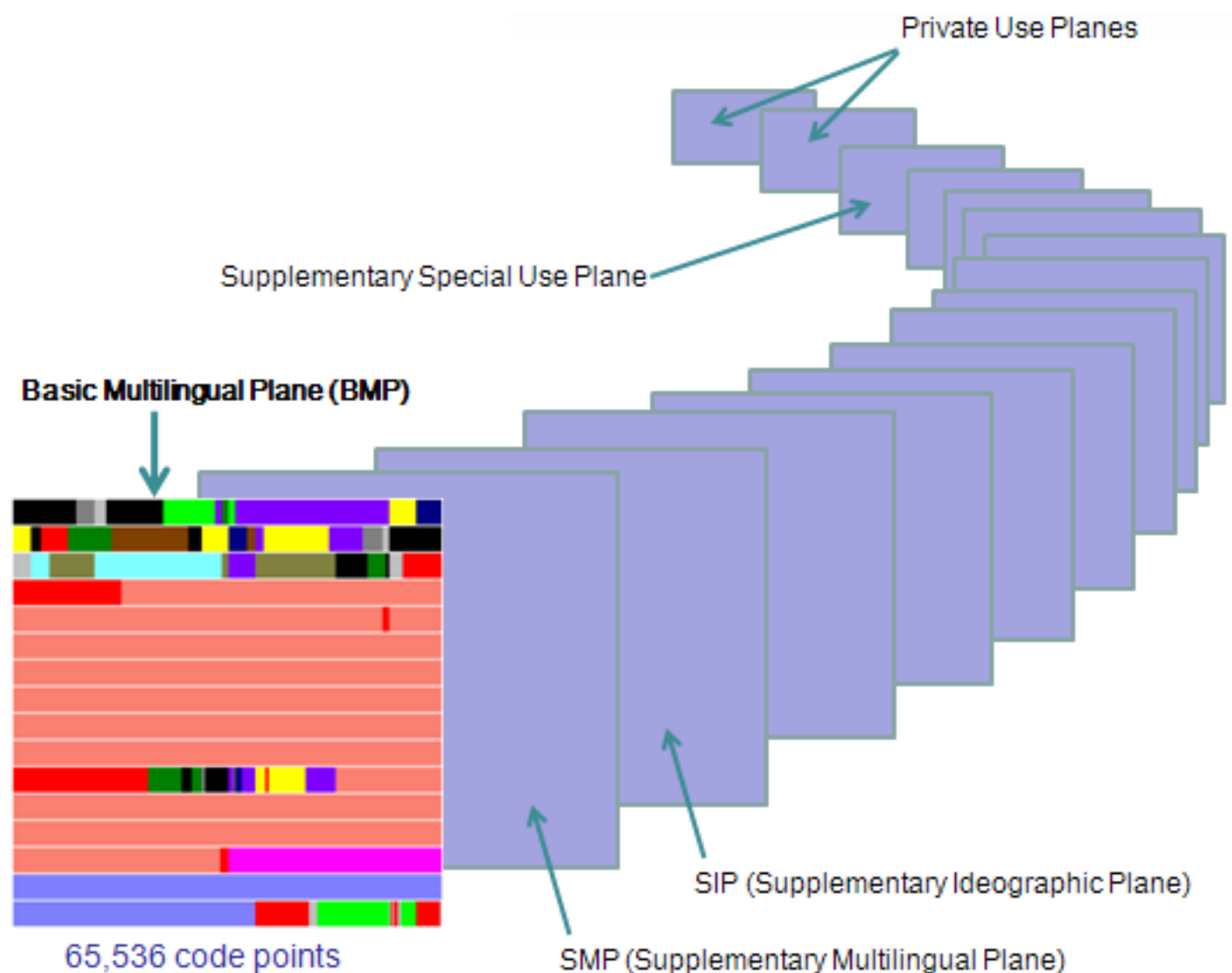
```

/* greek-ext */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'), url(http://fonts.gstatic.com/s/roboto/v14/-
2n2p-_Y08sg57CNwQfKNvesZW2x0Q-xsNq047m55DA.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/u0T0pm082MNkS5K0Q4rhqvesZW2x0Q-xsNq047m55DA.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-noMoG0miPesZW2x0Q-xsNq047m55DA.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/Fcx7Wwv80zT71A3E1X0AjvesZW2x0Q-xsNq047m55DA.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.



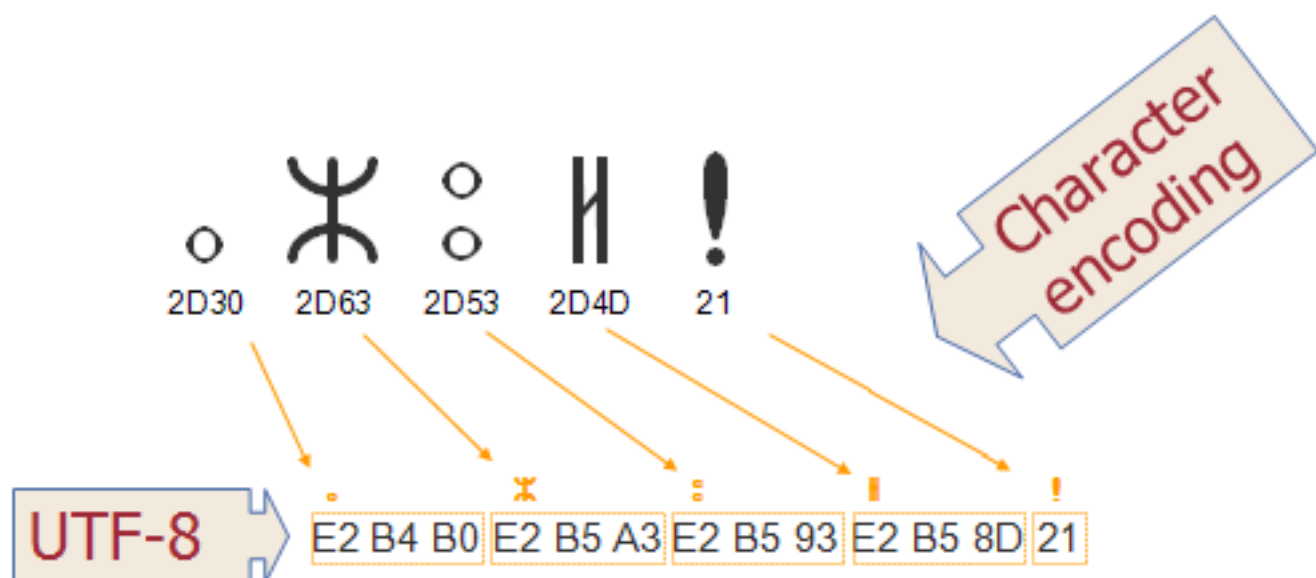
UTF-8 نجات بخش می‌شود

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:

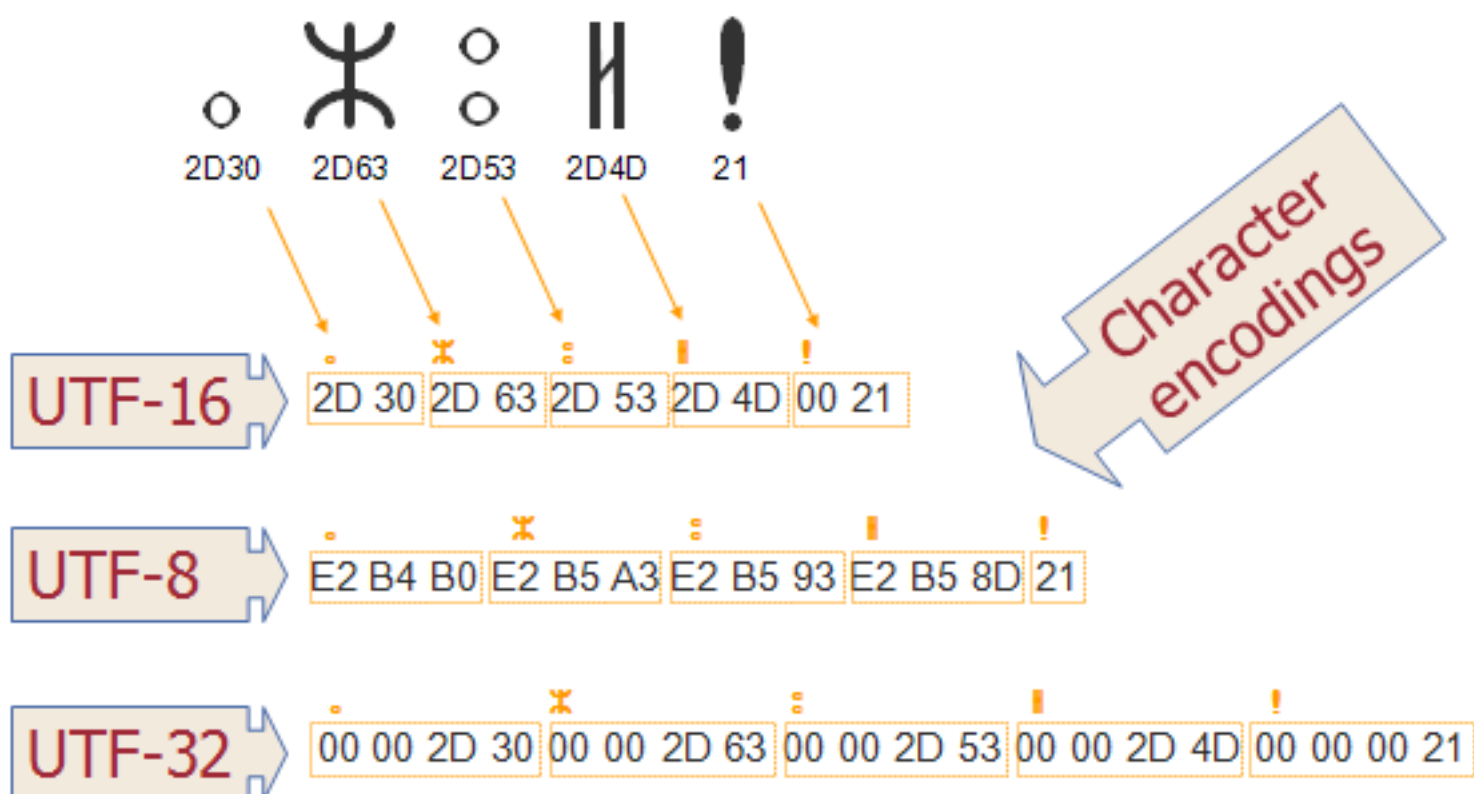
بسیاری از نرم افزارها و پروتکل‌ها هنوز 8 بیتی کار می‌کنند.

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، دو بایت 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 مثلاً از کدهای بین 0 تا 128 استفاده می‌کند و فرمت ذخیره آن UTF-8 باشد به ازای هر کاراکتر 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.

آشنایی با RLE

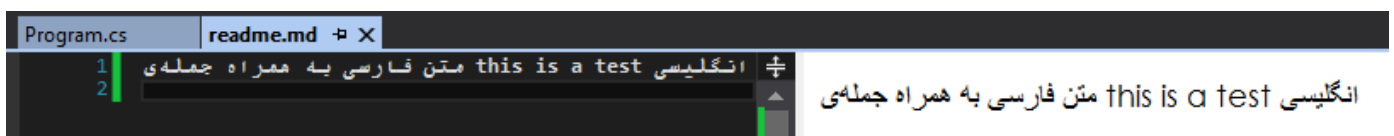
[الگوریتم پردازش دوطرفه‌ی یونیکد](#)، جهت و سمت نمایش متن را بر اساس خواص جهتی هر حرف مشخص می‌کند. در این حالت اگر متن مورد نمایش، انگلیسی و یا فارسی خالص باشند به خوبی عمل می‌کند؛ اما اگر ترکیب این دو را در یک رشته داشته باشیم، نیاز است نحوه‌ی جهت گیری و نمایش حروف را به Unicode bidirectional algorithm 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.

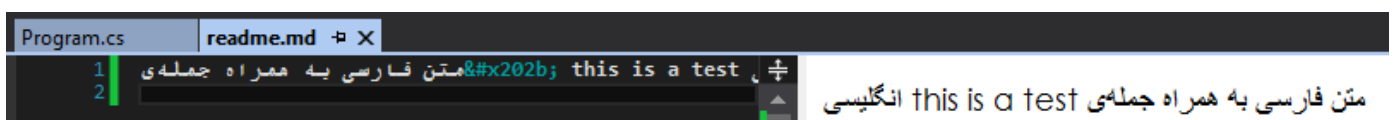
مثال اول

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:



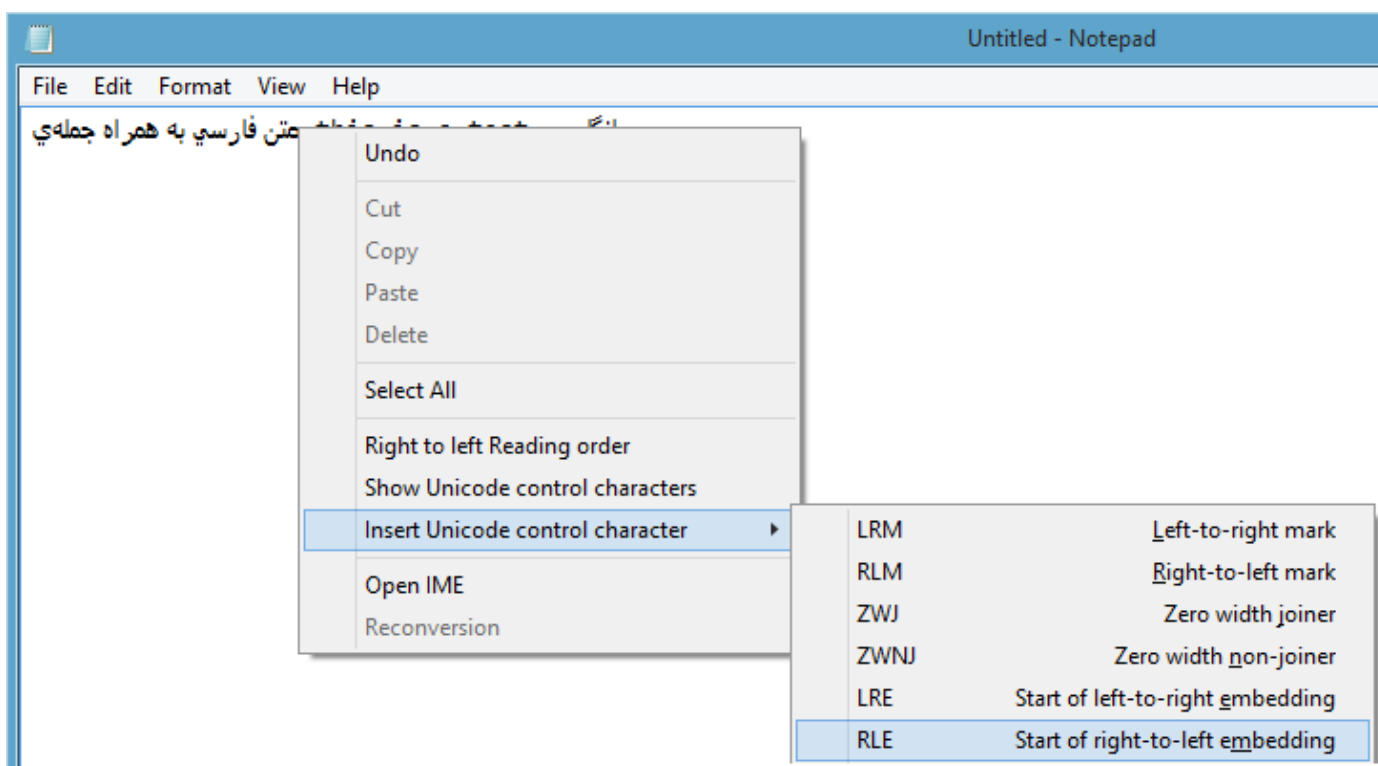
همانطور که ملاحظه می‌کنید، جمله معکوس شده است. برای رفع این مشکل می‌توان از کاراکتر نامرئی یونیکد 202b استفاده کرد. البته در mark down امکان تعریف ساده‌تر این کاراکتر به صورت ذیل نیز پیش بینی شده است:

```
&#x202b;
```

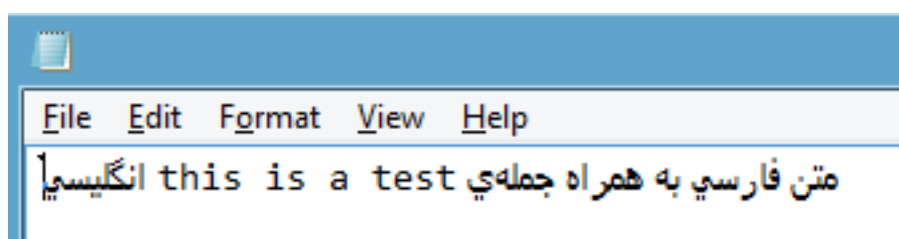


مثال دوم

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 نت 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 خصوصاً 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

تاریخ: 18/11/1390
شماره پروژه: 56/4/3/2/1
اسلش: A/13/12
بك اسلش: 14\13\12
مساوي و جمع: 5=3+2
سمي كولون: ;1+1=2
دلار: \$12
كاما: 12,34,67
نقطه: 12.34
پرانتز: متن (ساده)

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 اصلاً برای محیط‌های راست به چپ طراحی نشده‌است.

کار با اعراب در متون راست به چپ

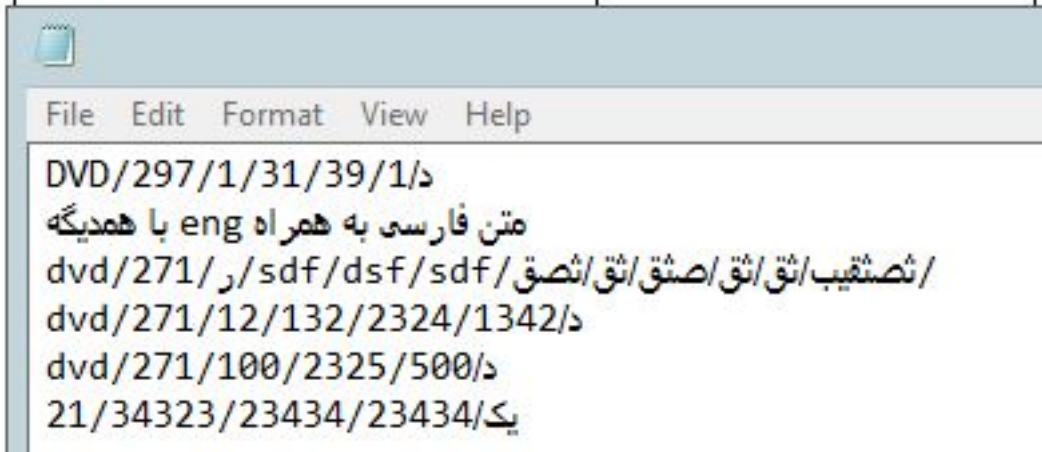
در یونیکد یک حرف می‌تواند از یک یا چند code point تشکیل شود. در حالت FormC، هر حرف، با اعراب آن یک code point را تشکیل می‌دهند. در حالت FormD، حرف با اعراب آن دو code point را تشکیل خواهند داد. به همین جهت نیاز است رشته را تبدیل به حالت D کرد تا بتوان اعراب آن‌را مجزای از حروف پایه، حذف نمود.
البته اعراب 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: ۱۷:۳۸ ۱۳۹۳/۱۰/۱۶

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



من میخوام مثل فایل ورد باشه و همه چیز سرجاش. اما وقتی همون فایل ورد کپی می‌گیرم داخل Notepad به هم می‌زنه. 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 پیدااش کنم.

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

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.IsNullOrEmpty(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، دقیقاً 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 قبلاً 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)