Форматни проблеми при многоезичен интерфейс за .net приложения

Д-р Мария Георгиева

Важен момент в програмирането на съвременни приложения е разработването ма многоезичен интерфейс, позволяващ на потребителя да работи на разбираем за него език. Това изисква по време на самото изпълнение (runtime) на приложението да се променя езикът на интерфейса. В .net този процес е доста улеснен с помощта на System.Globalization и принадлежащите 20-на класа към това пространство. Глобализираното приложение поддържа локализиран потребителски интерфейс и трябва да бъде езиково и културно неутрално. То коректно разпознава, обработва и визуализира характерните за дадена езикова култура формати – числа, дати и т.н. От своя страна локализацията е процес на конфигуриране на приложението за дадена езикова култура – комбинация от езика и мястото, където се говори. Например:

en-US - английски език; страна - USA
en-GB - английски език; страна – Great Britain
bg- BG – български език; страна – България
.net платформата предлага сравнително лесна реализация на този процес. Достатъчно е само да се укаже за дадена форма, че свойството Localizable е вярно, след което може да се промени езикът по подразбиране на формата (например *en-US* , да се смени с *bg- BG, fr-FR)* . Това води до създаване на ресурси за съответния език, които са разположени в поддиректории на директорията на самото приложение. Налице са няколко характерни особености за този процес:

- Контролите могат да са разположени на различни места в различните езикови варианти на формата
- Техните размери могат да са различни (често надписите на бутоните за различните езици имат различна дължина)
- Нови контроли могат да се добавят само във варианта за езика по подразбиране, след което се превеждат наименованията им и в останалите езикови форми на интерфейса (по принцип формата е една, но към нея има различни ресурсни файлове)

За да работим с различни езикови версии, трябва да включим :

C #	VB .net
`using System.Globalization;`	`Imports System.Threading`

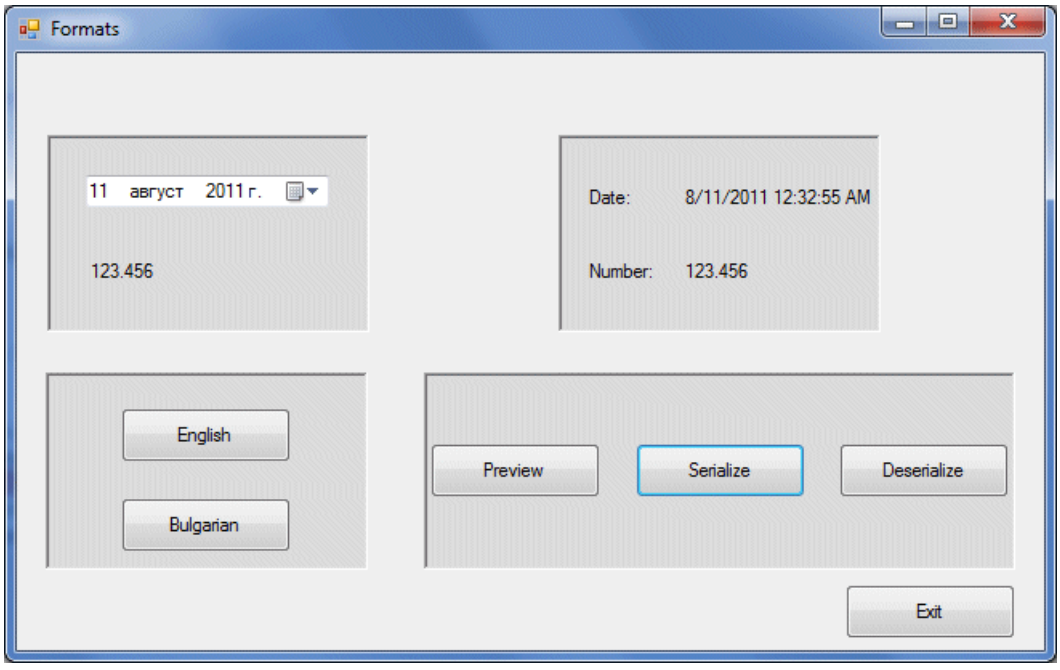
`using System.Threading;`	`Imports System.Globalization`

Самата промяна се осъществява с помощта на смяна на езиковата култура:

```
private void ChangeLanguage(string lang)
        {

            foreach (Control c in this.Controls)
            {
                Thread.CurrentThread.CurrentCulture = new CultureInfo(lang);
                Thread.CurrentThread.CurrentUICulture = new CultureInfo(lang);
                ComponentResourceManager resources = new
ComponentResourceManager(typeof(Form1));

                resources.ApplyResources(c, c.Name, new CultureInfo(lang));


            }

        }
```

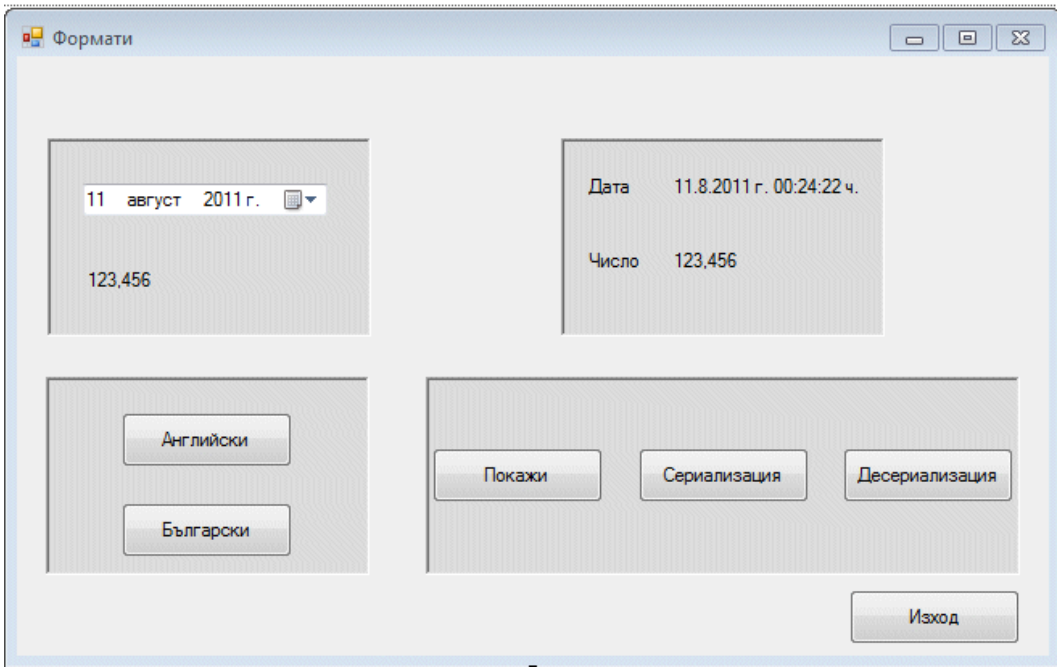
В този случай променливата **`lang`** може да има стойност ***bg- BG***, например. Методът позволява да се променят текстовете на контролите във формата. Основен момент в него е използването на класа **`CultureInfo,`** съдържащ информация за езика, формата на датите (къс, дълъг формат, разделител, подредба), формата на числата, валутата и т.н.

Ето например как изглежда интерфейса на едно просто приложение, създадено за онагледяване:



Данните в десния панел визуализират стойностите на DataTimePicker-а от лявата страна и обикновено десетично число.

С помощта на бутоните “English” и “Bulgarian” може да се сменя езикът на интерфейса:



Използването на различните езикови версии не създава никакви проблеми до момента, в който пожелаем да експортираме данни, които да бъдат ползвани след това – например в база данни, или както е в примера – със сериализация и запис във файл.

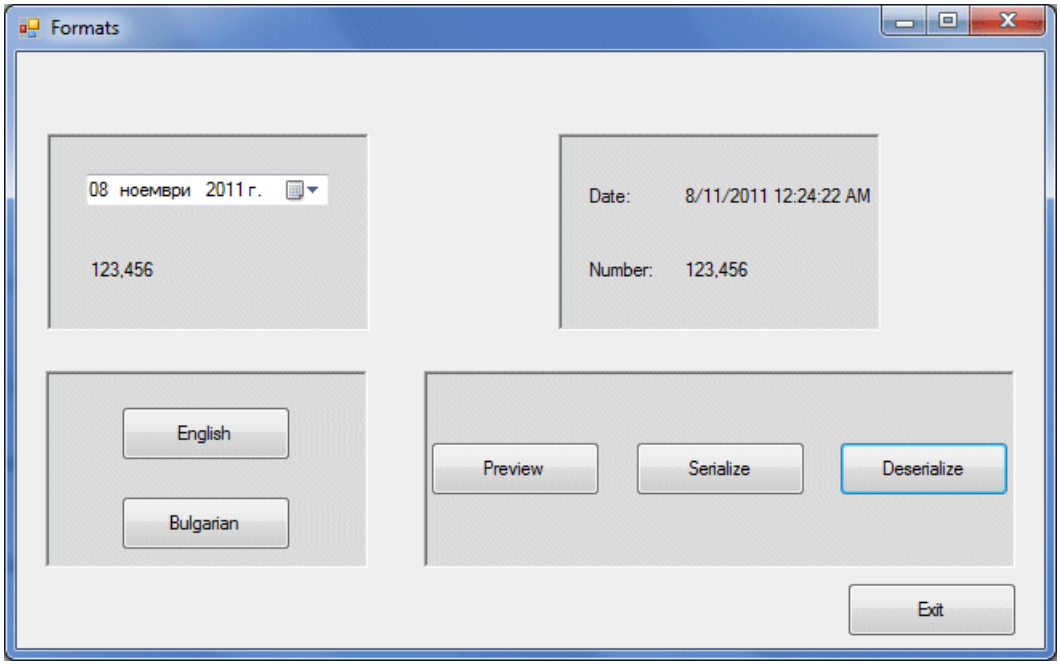
За целта беше използван клас

```
public class FormatExamp
    {
        private string mDate;
        private double mNumber;

        {
            get { return mDate; }
            set { mDate = value; }
        }
        public double MyNumber
        {
            get { return mNumber; }
            set { mNumber = value; }
        }

    }
```

Езикът беше установен на български и във файл след сериализацията беше записано

```
<?xml version="1.0"?>
<FormatExamp xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <MyDate>8.11.2011 г. 00:24:22 ч.</MyDate>
  <MyNumber>123.456</MyNumber>
</FormatExamp>
```

След което езикът на интерфейса (т.е. езиковата култура) беше сменен на английски и беше десериализиран файла. Данните от него бяха присвоени на DataTimePicker и етикета. И датата изведнъж се оказва не 11 август 2011, а 08 ноември 2011, тъй като форматът за *en-US* е mm/dd/year, а за *bg- BG* – dd.mm.year



Проблем бихме имали и при формата на реални числа, записани и четени с различни езикови култури:

```
1;350;0,1087;
2;350,12;0,6177;
3;350,24;0,2765;
4;350,36;0,417;
5;350,48;0,0526;
6;350,6;0,3615;
7;350,72;0,951;
8;350,84;0,8294;
9;350,96;0,8578;
10;351,08;0,131;
11;351,2;0,6442;
```

```
1;350;0.2529;
2;350.12;0.4018;
3;350.24;0.3943;
4;350.36;0.2913;
5;350.48;0.9837;
6;350.6;0.0589;
7;350.72;0.8452;
8;350.84;0.0344;
9;350.96;0.0043;
10;351.08;0.5277;
11;351.2;0.3198;
```

От горното изображение, представляващо запис на две различни измервания на спектър (номер на пиксел, дължина на вълната и стойност на интензитет) ясно се вижда влиянието на езиковата култура.

Четенето на такъв файл при интерфейс с различна езикова култура би довело до проблеми във визуализацията на спектъра.

Тези форматни проблеми могат лесно да се избегнат, ако се предвиди използването на информацията за текущата езикова култура.

За малкото приложение по-горе въпросът се решава, като се въведе допълнителна променлива

```
public class FormatExamp
    {
        private string mDate;
        private double mNumber;
        private string mLang;

        {
            get { return mDate; }
            set { mDate = value; }
        }
        public double MyNumber
        {
            get { return mNumber; }
            set { mNumber = value; }
        }
        public string MyLang
        {
            get{return mLang;}
            set { mLang = value; }
        }

    }
```

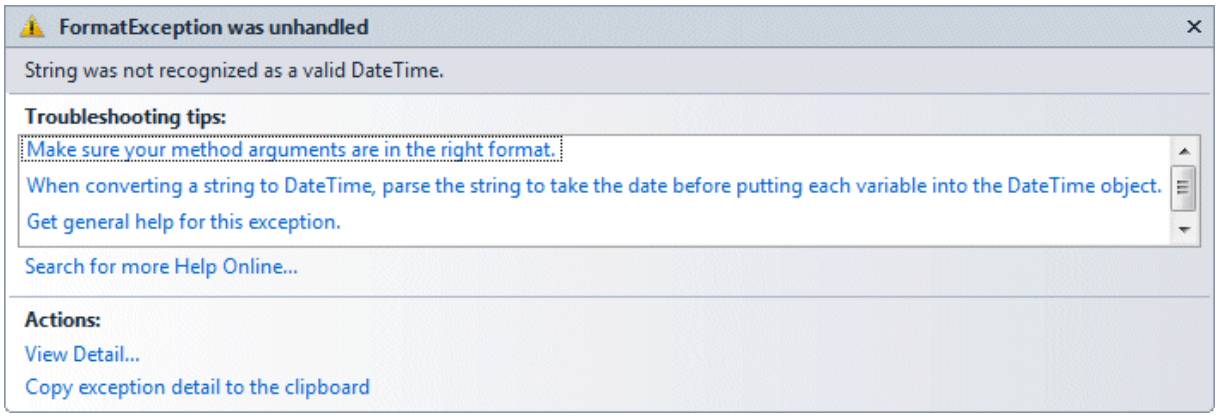
Промените в кода , които ще осигурят правилното възпроизвеждане на датата ще бъдат:

```
FormatExamp  newPers =(FormatExamp)mySer.Deserialize(inStream[1]);


// dateTimePicker1.Value = DateTime.Parse(newPers.MyDate);
              dateTimePicker1.Value = DateTime.Parse(newPers.MyDate, new
System.Globalization.CultureInfo(newPers.MyLang));
```

т.е. допълнителната информация, която въведохме и която е записана също във файла, позволява винаги правилно да се форматира информацията, съобразно езика на който е създадена, независимо от текущата култура

Коментираният ред (маркиран в зелено) води до обръщане на стойностите на месеца и деня. В примера програмата продължи изпълнението си, но при дата 25.07.2011, например, би спряла изпълнението си със съобщение за грешка

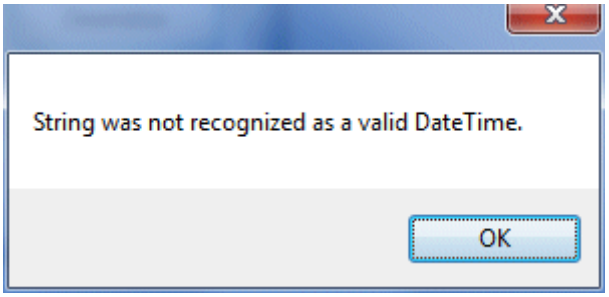
[1] От тип FileStream



Затова е препоръчително, винаги когато имаме подобни конвертирания, да се опитаме да отработим прекъсването

```
try
{
    dateTimePicker1.Value = DateTime.Parse(newPers.MyDate);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

В този случай ще получим съобщението:



но програмата ще продължи изпълнението.

От кратките примери по-горе става ясно, че създаването на многоезични приложения в .net е много лесно, стига програмистите да се съобразяват с възможните форматни проблеми.

Литература

Liberty, J. (2005). *Programming C#, 4th Edition.* O'Reilly.

Deitel. (2005). *Visual C#® 2005: How to Program, Second Edition.* Prentice Hall.