Module 19

"Globalization"

TEKNOLOGISK INSTITUT

Agenda

- **Globalization and Localization**
- Formatting Data for Globalization

Introducing Globalization

- Globalization
  - Preparing the application to be compliant with local cultures
- Localization
  - Creating resources for a specific given culture

- These terms are (sloppily ☺) used interchangeably to mean approximately the same thing

- Localized assemblies
  - Are known as ”satellite assemblies”
  - Can be deployed after main application
  - Each framework usually provide a separate mechanisms for localization

The **CultureInfo** Class

- Cultures consist of two elements
  - Language
    - Captured by two-letter language code, e.g. ″**da**″, ″**en**″, **de**″
  - Country or region
    - Captured by two-letter region code, e.g. ″**DK**″, ″**US**″, ″**DE**″
- Specific cultures specify both language and region codes
  - ″**da-DK**″, ″**en-US**″, ″**es-US**″, …
- Neutral cultures specify only language codes
- Cultures are captured by **CultureInfo** objects

```
CultureInfo ci = new CultureInfo( "da-DK" );
```

Setting and Enumerating Cultures

- You can set the underlying default **`CultureInfo`** objects via
  - `Thread.CurrentThread.CurrentCulture`
    - Controls formatting of numbers, dates and times, …
    - Cannot be set to neutral cultures!
  - `Thread.CurrentThread.CurrentUICulture`
    - Determines which resources the resource manager loads

- **`CultureInfo.GetCultures()`** enumerates cultures
  - `CultureTypes.AllCultures`
  - `CultureTypes.SpecificCultures`
  - `CultureTypes.NeutralCultures`
  - …

The **RegionInfo** Class

- The **`RegionInfo`** class
  - Captures info about a specific country of region
  - Contains regional formatting information
    - Localized country name
    - Localized and neutral currency name and formatting
    - Metric System information
    - …
  - Can be created as
    - Neutral
    - Specific

```
CultureInfo culture =
   new CultureInfo( "es-US", true ); // Override

RegionInfo region1 = new RegionInfo( "US" );
RegionInfo region2 = new RegionInfo( "es-US" );
```

- User overriding
  - **`CultureInfo`** can be user overridden
  - **`RegionInfo`** cannot be user overridden

Agenda

- Globalization and Localization
- **Formatting Data for Globalization**

Introducing `CultureInfo` Formatting

- The `CultureInfo` class has additional formatting helper classes accessible through properties

`NumberFormatInfo`	`CultureInfo.NumberFormat`
`DateTimeFormatInfo`	`CultureInfo.DateTimeFormat`
`CompareInfo`	`CultureInfo.CompareInfo`
…	

- In the following, we will cover each helper class

NumberFormatInfo Class

- Both **NumberFormatInfo** and **CultureInfo** implement **IFormatProvider**
  - Pass either to supported methods
    - ToString(), Parse(), and TryParse()

```
CultureInfo ci = new CultureInfo("en-US");
NumberFormatInfo nfi = ci.NumberFormat;
string number = 99999.ToString( "N2", nfi );
Console.WriteLine( number );
```

```
double value;
if( double.TryParse( "99,999.00", NumberStyles.Any, nfi,
                     out value ) )
   Console.WriteLine( "Successfully parsed double" );
else
   Console.WriteLine( "Could not parse double from string"
```

`NumberFormatInfo` Properties

- `NumberFormatInfo` properties
  - `CurrentInfo`
  - `CurrencyPositivePattern`
  - `CurrencyNegativePattern`
  - `CurrencySymbol`
  - `CurrencyDecimalSeparator`
  - `PositiveSign`
  - `NegativeSign`
  - …

- Don’t try to memorize all these properties! ☺

DateTimeFormatInfo Class

- Both **DateTimeFormatInfo** and **CultureInfo** implement **IFormatProvider**
  - Pass either to supported methods
    - **ToString()**, **Parse()**, **TryParse()**, and **TryParseExact()**

```
CultureInfo ci = new CultureInfo("en-US");
DateTimeFormatInfo dtfi = ci.DateTimeFormat;
DateTime now = DateTime.Now;
Console.WriteLine( now.ToString( "D", dtfi ) );
```

```
DateTime value;
if( DateTime.TryParse( "10-19-1986", dtfi, DateTimeStyles.None,
               out value ) )
   Console.WriteLine( "Successfully parsed DateTime from string" );
else
   Console.WriteLine( "Could not parse DateTime from string" );
```

DateTimeFormatInfo Methods and Properties

TEKNOLOGISK INSTITUT

- `DateTimeFormatInfo` properties
  - `CurrentInfo`
  - `Calendar`
  - `DateSeparator`
  - `LongDatePattern`
  - `LongTimePattern`
  - …
- `DateTimeFormatInfo` methods
  - `GetAbbreviatedDayName()`
  - `GetAbbreviatedMonthName()`
  - …

- Don’t try to memorize all these properties! ☺

CompareInfo Class

- The **CompareInfo** class facilitates culture-specific comparison of strings

```
CultureInfo ci = new CultureInfo( "da-DK" );
CompareInfo compareInfo = ci.CompareInfo;
Console.WriteLine( compareInfo.Compare(
   "Aarhus", "Århus", CompareOptions.IgnoreCase )
);
```

- Note: Used differently from
  - `NumberFormatInfo`
  - `DateTimeFormatInfo`

Comparing and Sorting Objects

TEKNOLOGISK INSTITUT

- By default items are sorted as specified in the default culture

```
string[] places = { "Aalborg", "Århus", "Aabybro", "Nørre Aaby" };
Array.Sort( places );
```

- Sorting can be performed in an explicit culture-sensitive manner using the **`StringComparer`** class

```
string[] places = { "Aalborg", "Århus", "Aabybro", "Nørre Aaby" };
Array.Sort( places, StringComparer.Create( culture, true ) );
foreach( string name in places )
{
   Console.WriteLine( name );
}
```

Invariant Culture

- Invariant culture
  - Culture-insensitive
  - Associated with the English language and no specific region
  - Obtained via **`CultureInfo.InvariantCulture`**
- Alternatively for sorting
  - `StringComparer.InvariantCulture` or
  - `StringComparer.InvariantCultureIgnoreCase`

```
string[] places = { "Aalborg", "Århus", "Aabybro", "Nørre Aaby" };
Array.Sort( places, StringComparer.InvariantCultureIgnoreCase );
foreach( string name in places )
{
   Console.WriteLine( name );
}
```

Building Custom Cultures

- Use the **`CultureAndRegionInfoBuilder`** class to build and install custom cultures
  - In the **`System.Globalization`** namespace
    - Add reference to **`sysglobl.dll`**!

- `CultureAndRegionInfoBuilder` methods
  - `LoadDataFromCultureInfo()`
  - `LoadDataFromRegionInfo()`
  - `Save()`
    - Save custom culture to LDML file
  - `CreateFromLdml()` static
  - `Register()`
    - Requires administrative rights
  - `Unregister()`
    - Requires administrative rights

Summary

- Globalization and Localization
- Formatting Data for Globalization

Question

- You are developing an inventory report system for a customer based in the United States. The customer has a local office in Denmark. You must ensure that when users in the local office generate a report, they will see the date displayed in danish format.

  Which code segment should you use?

Question (Cont'd)

a)

```
string date = DateTime.Today.Month.ToString( "da-DK" );
```

b)

```
string date = DateTimeFormatInfo.CurrentInfo.GetMonthName(
   DateTime.Today.Month );
```

c)

```
DateTimeFormatInfo dtfi =
   new CultureInfo( "da-DK", false ).DateTimeFormat;
DateTime dt = new DateTime( DateTime.Today.Year,
                            DateTime.Today.Month,
                            DateTime.Today.Day );
string date = dt.ToString(dtfi.LongDatePattern);
```

d)

```
Calendar cal = new CultureInfo("da-DK", false).Calendar;
DateTime dt = new DateTime(
   DateTime.Today.Year,  DateTime.Today.Month, DateTime.Today.Day );
string date = dt.ToString();
```

TEKNOLOGISK
INSTITUT