

The Timer and Time

Agenda

- Timers
- The DateTime Structure

4 Different Timers

- There are 4 Timer classes in the .NET framework:
 - System.Windows.Forms.Timer (*not relevant for WPF developers*)
 - System.Timers.Timer
 - System.Threading.Timer
 - System.Windows.Threading.DispatcherTimer
- And they have different capabilities

System.Threading.Timer

- Uses a TimerCallback delegate to specify the methods associated with a Timer.
- **The method (the callback) execute in a separate thread that is automatically allocated by the system → real multithreading!**
- When creating a timer, the application specifies an amount of time to wait before the first invocation of the delegate methods (due time)
- And an amount of time to wait between subsequent invocations (period).
- A timer invokes its methods when its due time elapses, and invokes its methods once per period thereafter.
- You can change these values, or you can disable the timer, by using the Change method.
- When a timer is no longer needed, use the Dispose method to free the resources held by the timer.

General-purpose
multithreaded timer

System.Timers.Timer

- "The Server Timer"
- The server-based timer is designed for use with **worker threads** in a **multi-threaded** environment.
- Because they use a different architecture, server-based timers have the potential to be much more accurate than Windows timers.
- Use this timer in Non - Windows Forms / WPF apps.
- *This timer simply wraps the System.Threading.Timer, providing additional convenience while using the identical underlying engine.*

General-purpose
multithreaded timer

Server Timer Example

```
using System.Timers;
```

```
public class Timer1  
{
```

```
    public static void Main()  
    {
```

```
        System.Timers.Timer aTimer = new System.Timers.Timer();  
        aTimer.Elapsed += new ElapsedEventHandler(OnTimedEvent);  
        // Set the Interval to 3 seconds.
```

```
        aTimer.Interval=3000;
```

```
        aTimer.Enabled=true;
```

```
        Console.WriteLine("Press \'q\' to quit the sample.");
```

```
        while(Console.Read()!='q');
```

```
    }
```

```
    //Specify what you want to happen when the Elapsed event is raised.
```

```
    public static void OnTimedEvent(object source, ElapsedEventArgs e)
```

```
    {
```

```
        Console.WriteLine("Hello world!");
```

```
    }
```

```
}
```

The DispatcherTimer

- Is like System.Timers.Timer in the members that it expose (Interval, Tick, Start, and Stop).
 - But differ in how it work internally.
- Instead of using the thread pool to generate timer events, the WPF timer rely on the message pumping mechanism of their underlying user interface model.
 - This means that the Tick event always fires on the same thread that originally created the timer.
- This has a number of benefits:
 - You can forget about thread safety.
 - A fresh Tick will never fire until the previous Tick has finished processing.
 - **You can update user interface elements and controls directly from Tick event handling code, without calling Control.Invoke or Dispatcher.Invoke.**
 - It sounds too good to be true, until you realize that a program employing these timers is not really multithreaded—**there is no parallel execution.** One thread serves all timers—as well as the processing UI events.



For WPF

DispatcherTimer Disadvantages

This brings us to the disadvantage of single-threaded timers:

- Unless the Tick event handler executes quickly, the user interface becomes unresponsive.
 - This makes the WPF timers suitable for only small jobs, typically those that involve updating some aspect of the user interface (e.g., a clock or countdown display).
 - Otherwise, you need a multithreaded timer.
- In terms of precision, the single-threaded timers are similar to the multithreaded timers (tens of milliseconds),
 - although they are typically less **accurate**, because they can be delayed while other user interface requests (or other timer events) are processed.

DispatcherTimer Demo

```
partial class MyWindow : Window {
    DispatcherTimer dt;
    public MyWindow( ) {
        dt = new DispatcherTimer( );
        dt.Tick += dt_Tick;
        dt.Interval = TimeSpan.FromSeconds(2);
        dt.Start( );
    }
    Random rnd = new Random( );
    void dt_Tick(object sender, EventArgs e) {
        byte[] vals = new byte[3];
        rnd.NextBytes(vals);
        Color c = Color.FromRgb(vals[0], vals[1], vals[2]);

        // OK to touch UI elements, as the DispatcherTimer
        // calls us back on the UI thread
        this.Background = new SolidColorBrush(c);
    }
}
```

DATE AND TIME

The DateTime Structure

- The **DateTime** value type represents dates and times with values ranging
 - from 12:00:00 midnight, January 1, 0001 C.E. (Common Era)
 - to 11:59:59 P.M., December 31, 9999 C.E.
- Time values are measured in 100-nanosecond units called ticks.
- A particular date is the number of ticks since 12:00 midnight, January 1, 1 C.E. in the Gregorian Calendar.
- A ticks value of 312413760000000000L represents the date, Friday, January 01, 0100 12:00:00 midnight.
- A **DateTime** value is always expressed in the context of an explicit or default calendar.

```
Console.WriteLine("Dags dato: " + DateTime.Now);  
Console.WriteLine("Dags dato: " + DateTime.Now.ToString("u"));  
  
DateTime date1 = new DateTime(2012, 9, 24, 8, 30, 52);  
string dateString = "23/9/2012 12:01:52";  
DateTime date2 = DateTime.Parse(dateString);
```

TimeSpan

- A TimeSpan represents a time interval.
- This means, for example, that you can subtract one instance of DateTime from another to obtain the time interval (~ TimeSpan) between them.
- Or you could add a positive TimeSpan to the current DateTime to calculate a future date.
- Time values can be negative or positive, and expressed in units such as ticks, seconds, or instances of TimeSpan.

```
DateTime date1 = new DateTime(2012, 9, 24, 8, 30, 52);  
string dateString = "23/9/2012 12:01:52";  
DateTime date2 = DateTime.Parse(dateString);  
TimeSpan diff = date1 - date2;  
Console.WriteLine("Timespan: " + diff);
```

Coordinated Universal Time

- Coordinated universal time (UTC) was previously known as Greenwich Mean Time (GMT).
- Local time is the date and time on the computer you are using.
- Offset is the difference between local time and UTC.
 - $\text{local time} = \text{UTC} + \text{offset}$
- TimeSpan **GetUtcOffset**(DateTime *time*);
 - *time* must be in the Gregorian calendar and the time zone represented by this instance.
- For example, in the United States Pacific Standard time zone, which has -8 hours of offset,
`GetUtcOffset(new DateTime(1999, 1, 1))`
returns `-2880000000000`.

Calendar

- A calendar divides time into measures, such as weeks, months, and years. The number, length, and start of the divisions vary in each calendar.
- Any moment in time can be represented as a set of numeric values using a particular calendar.
- An implementation of Calendar can map any DateTime value to a similar set of numeric values.
- In order to make up for the difference between the calendar year and the actual time that the earth rotates around the sun or the actual time that the moon rotates around the earth, a leap year has a different number of days than a standard calendar year.
- Each Calendar implementation defines leap years differently.

Calendars

- The System.Globalization namespace includes the following Calendar implementations:

ChineseLunisolarCalendar
EastAsianLunisolarCalendar
GregorianCalendar
HebrewCalendar
HijriCalendar
JapaneseCalendar
JapaneseLunisolarCalendar
JulianCalendar
KoreanCalendar
KoreanLunisolarCalendar

PersianCalendar
TaiwanCalendar
TaiwanLunisolarCalendar
ThaiBuddhistCalendar
UmAlQuraCalendar

DateTime Format Specifiers

Format specifier	Associated Property/Description
d	ShortDatePattern
D	LongDatePattern
f	Full date and time (long date and short time)
F	FullDateTimePattern (long date and long time)
g	General (short date and short time)
G	General (short date and long time)
m, M	MonthDayPattern
o	Round-trip date/time pattern
r, R	RFC1123Pattern
s	SortableDateTimePattern (based on ISO 8601) using local time
t	ShortTimePattern
T	LongTimePattern
u	UniversalSortableDateTimePattern using the format for universal time display
U	Full date and time (long date and long time) using universal time
y, Y	YearMonthPattern

Custom DateTime Format Specifiers

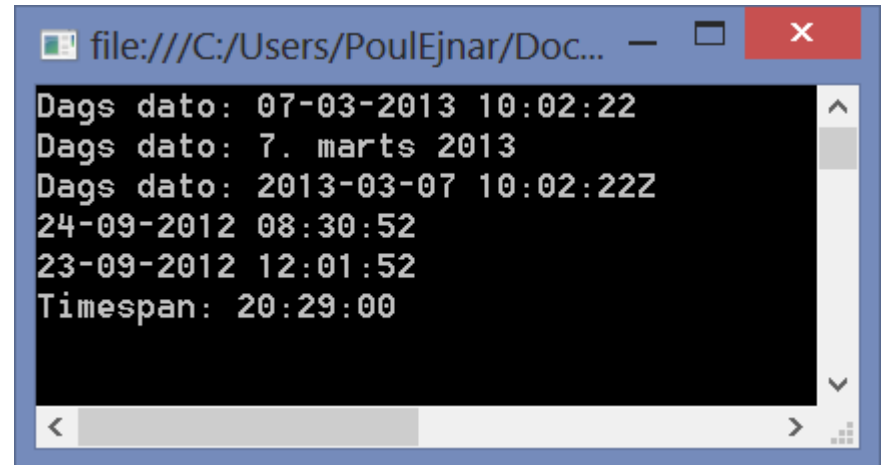
Format specifier	Associated Property/Description
d	Represents the day of the month as a number from 1 through 31. A single-digit day is formatted without a leading zero.
dd	Represents the day of the month as a number from 01 through 31. A single-digit day is formatted with a leading zero.
ddd	Represents the abbreviated name of the day of the week as defined in the current <code>System.Globalization.DateTimeFormatInfo.AbbreviatedDayNames</code> property.
dddd (plus any number of additional "d" specifiers)	Represents the full name of the day of the week as defined in the current <code>System.Globalization.DateTimeFormatInfo.DayNames</code> property. <i>Only an extract – there are many more specifiers – use online help!</i>

A custom DateTime format string consists of two or more characters. For example, if the format string consists only of the specifier “d”, the format string is interpreted as a standard DateTime format string.

To use a single custom DateTime format specifier, include a space before or after the DateTime specifier, or include a percent (%) format specifier before the DateTime specifier. For example, the format strings “h ” and “%h” are interpreted as custom DateTime format strings that display the hour represented by the current DateTime object.

DateTime Demo

```
Console.WriteLine("Dags dato: " + DateTime.Now);  
Console.WriteLine("Dags dato: " + DateTime.Now.ToLongDateString());  
Console.WriteLine("Dags dato: " + DateTime.Now.ToString("u"));  
  
DateTime date1 = new DateTime(2012, 9, 24, 8, 30, 52);  
string dateString = "23/9/2012 12:01:52";  
DateTime date2 = DateTime.Parse(dateString);  
Console.WriteLine(date1);  
Console.WriteLine(date2);  
TimeSpan diff = date1 - date2;  
Console.WriteLine("Timespan: " + diff);
```



A screenshot of a Windows command prompt window. The title bar shows the file path "file:///C:/Users/PoulEjnar/Doc...". The command prompt displays the output of the C# code: "Dags dato: 07-03-2013 10:02:22", "Dags dato: 7. marts 2013", "Dags dato: 2013-03-07 10:02:22Z", "24-09-2012 08:30:52", "23-09-2012 12:01:52", and "Timespan: 20:29:00".

References & Links

- If you want to develop a calendar app you may find help is this project:

<http://www.codeproject.com/Articles/168662/Time-Period-Library-for-NET>