Working with Dates and Times

by Steve Smith

The DateTime Type

C# has built-in support for dates and times in the form of the `System.DateTime` type. This type can be used to represent a date and time, and it provides static access to the system clock which can be used to get the current time and/or date. The type also supports many operations that can make it easy to work with and manipulate dates and times.

The following block demonstrates some `DateTime` operations:

```
var currentTime = DateTime.Now; // current time
var today = DateTime.Today; // current date - time is midnight
var someDate = new DateTime(2016,7,1); // 1 July 2016, midnight
var someMoment = new DateTime(2016,7,1,8,0,0); // 1 July 2016, 08:00.00
var tomorrow = DateTime.Today.AddDays(1);
var yesterday = DateTime.Today.AddDays(-1);
var someDay = DateTime.Parse("7/1/2016");
```

```
using System;

class Program
{
    static void Main()
    {
        var currentTime = DateTime.Now; // current time
        var today = DateTime.Today; // current date - time is midnight
        var someDate = new DateTime(2016,7,1); // 1 July 2016, midnight
        var someMoment = new DateTime(2016,7,1,8,0,0); // 1 July 2016, 08:00.00
        var tomorrow = DateTime.Today.AddDays(1);
        var yesterday = DateTime.Today.AddDays(-1);
        var someDay = DateTime.Parse("7/1/2016");

        Console.WriteLine($"{nameof(currentTime)}: {currentTime}");
        Console.WriteLine($"{nameof(today)}: {today}");
        Console.WriteLine($"{nameof(someDate)}: {someDate}");
        Console.WriteLine($"{nameof(someMoment)}: {someMoment}");
        Console.WriteLine($"{nameof(tomorrow)}: {tomorrow}");
        Console.WriteLine($"{nameof(yesterday)}: {yesterday}");
        Console.WriteLine($"{nameof(someDay)}: {someDay}");
    }
}
```

Each of the above methods creates an instance of a `DateTime` .

Tip {.tip .newLanguage }
The `DateTime.Now` and `DateTime.Today` properties are *static*, which means you can access them without having an instance of a `DateTime` . No matter where you access these properties, their value will be read from the current time according to the machine's system clock.

Tip {.tip .REPL}
You can use the C# `nameof` function to get a string representing the name of a C# construct.

Formatting Dates and Times

There are many ways to display dates and times. Different countries and regions have different preferred formats for displaying dates. Times can be represented using 12-hour AM/PM style times, or using 24-hour or "military" times. The DateTime type supports many different options for converting a value into a string, along with support for custom formats.

Learn more about custom date and time format strings.

Getting to Parts of Dates and Times

`DateTime` type has many properties you can use to access different individual components of the date or time, such as the month, day, year, hour, minute, etc. The following sample demonstrates some of these properties:

```
var someTime = new DateTime(2016,7,1,11,10,9); // 1 July 2016 11:10:09 AM
int year = someTime.Year; // 2016
int month = someTime.Month; // 7
int day = someTime.Day; // 1
int hour = someTime.Hour; // 11
int minute = someTime.Minute; // 10
int second = someTime.Second; // 9
```

```
using System;

class Program
{
    static void Main()
    {
        var someTime = new DateTime(2016,7,1,11,10,9); // 1 July 2016 11:10:09 AM
        int year = someTime.Year; // 2016
        int month = someTime.Month; // 7
        int day = someTime.Day; // 1
        int hour = someTime.Hour; // 11
        int minute = someTime.Minute; // 10
        int second = someTime.Second; // 9

        Console.WriteLine($"{nameof(someTime)}: {someTime}");
        Console.WriteLine($"{nameof(year)}: {year}");
        Console.WriteLine($"{nameof(month)}: {month}");
        Console.WriteLine($"{nameof(day)}: {day}");
```

```
        Console.WriteLine($"{nameof(hour)}: {hour}");
        Console.WriteLine($"{nameof(minute)}: {minute}");
        Console.WriteLine($"{nameof(second)}: {second}");
    }
}
```

Calculating Durations Between DateTimes

Sometimes it can be useful to think about the difference between two times, or the duration of an event. For instance, you could calculate how many days are left in the year. To do so, you would need to determine the first day of the next year, and then compare that to today. The following code shows how to do this:

```
DateTime nextYear = new DateTime(DateTime.Today.Year+1, 1, 1);
TimeSpan duration = nextYear - DateTime.Today;
Console.WriteLine($"There are {duration.TotalDays} days left in the year");
```

```
using System;

class Program
{
    static void Main()
    {
        DateTime nextYear = new DateTime(DateTime.Today.Year+1, 1, 1);
        TimeSpan duration = nextYear - DateTime.Today;
        Console.WriteLine($"There are {duration.TotalDays} days left in the year");
    }
}
```

The first line of code creates a new `DateTime` instance for January 1st of the current year, plus one. The second line takes this instance and *subtracts* the current date from it. This produces a new type called a `TimeSpan`, which is used to represent a span of time, or a duration. `TimeSpan` provides many ways to represent its value, depending on the units you want to use. In this case, the `TotalDays` property will show how many full days the `TimeSpan` includes.