

Copyright © 2015 by The Learning House.

All rights reserved. No part of these materials may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of The Learning House. For permission requests, write to The Learning House, addressed “Attention: Permissions Coordinator,” at the address below.

The Learning House
427 S. 4th Street #300
Louisville KY 40202

The DateTime Data Type

.NET Cohort

Coding Bootcamp

About DateTime

DateTime is a built-in *struct* in .NET that lives in the System namespace.

As the name suggests, it holds both date and time information. Dates can be from 01/01/0001 to 12/31/9999 and times can be any time with millisecond accuracy.

Since it is a struct, by default it cannot be null. When working with database fields that can be null, we have to mark it as nullable like this:

```
DateTime? nullableDate;  
nullableDate = null;
```

UTC Time

UTC (Co-ordinated Universal Time) is the world standard for defining time. Unlike local times which can vary (daylight savings time), a UTC time is the same everywhere on the planet.

When we build an application that uses time information, an important consideration is whether we want to store times in local time or UTC time.

A general rule of thumb is that if time information needs to be shown internationally in context, use UTC. If it will only be shown in your region, use local time.

Converting one to the other isn't a big deal, so we tend to default to local time for simplicity.

Ticks

In .NET, the smallest measure of time is a *tick*. A tick equals one ten-millionth of a second (100 ns).

When we work with a DateTime under the covers, it is all numeric math with any date/time being stored as the number of ticks since 01/01/0001.

This is why we can add and subtract from our DateTimes, as we will see later.

Creating a DateTime

To create a DateTime, we must use the DateTime *constructor* with the *new* keyword:

```
// 12-25-2013 (midnight)
DateTime xmas = new DateTime(2013, 12, 25);

// invalid, will cause error at run-time
DateTime error = new DateTime(2013, 13, 35);

// 12/25/2013 8:15:30 AM
DateTime morning = new DateTime(2013, 12, 25, 8, 15, 30);

// 12/25/2013 8:15:30:005 AM
DateTime afternoon = new DateTime(2013, 12, 25, 8, 15, 30, 5);
```

Our DateTime as Ticks

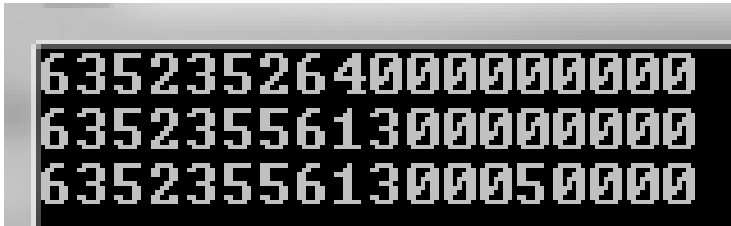
Let's print out the ticks of our DateTime variables

```
// 12-25-2013 (midnight)
DateTime xmas = new DateTime(2013, 12, 25);

// 12/25/2013 8:15:30 AM
DateTime morning = new DateTime(2013, 12, 25, 8, 15, 30);

// 12/25/2013 8:15:30:005 AM
DateTime afternoon = new DateTime(2013, 12, 25, 8, 15, 30, 5);

Console.WriteLine(xmas.Ticks);
Console.WriteLine(morning.Ticks);
Console.WriteLine(afternoon.Ticks);
```



```
6352352640000000000
6352355613000000000
6352355613000500000
```

String Parsing

When we read in dates from users, it can be a pain to ensure the correct format is used.

The `DateTime.Parse` and `DateTime.TryParse` method converts strings to `DateTime` values using the local date format.

```
DateTime xmasMidnight = DateTime.Parse("12/25/2013");  
  
DateTime xmasMorning;  
DateTime.TryParse("12/25/2013 8:15:30.006", out xmasMorning);
```


Comparisons

Because the underlying data of a DateTime is ticks since 01/01/0001, it's easy to compare and order DateTime values. In the back end, it is simply comparing the ticks.

```
DateTime d1 = DateTime.Parse("12/25/2013");  
DateTime d2 = DateTime.Parse("01/01/2014");  
  
bool result;  
  
result = d1 == d2; // false  
result = d1 != d2; // true  
result = d1 > d2; // false  
result = d1 < d2; // true;
```

Getting the Current Date

The DateTime structure has a few static methods to help get the current system DateTime:

```
DateTime d1;  
  
// current date to millisecond  
d1 = DateTime.Now;  
  
// current date (midnight)  
d1 = DateTime.Today;  
  
// current UTC format date/time  
d1 = DateTime.UtcNow;
```

Getting Parts of a DateTime

We can pull out specific pieces of a DateTime value simply by calling the properties:

```
DateTime d1 = DateTime.Now;  
  
int year = d1.Year;  
int month = d1.Month;  
int day = d1.Day;  
int hour = d1.Hour;  
int minute = d1.Minute;  
int second = d1.Second;  
int millisecond = d1.Millisecond;
```

DayOfWeek Property

The DateTime struct has a special DayOfWeek property that returns an enum of what day of the week a date falls on. It starts with DayOfWeek.Sunday:

```
DateTime d1 = DateTime.Now;

DayOfWeek day = d1.DayOfWeek;

switch (day)
{
    case DayOfWeek.Saturday:
    case DayOfWeek.Sunday:
        Console.WriteLine("It's the weekend!");
        break;
    default:
        Console.WriteLine("It's a weekday");
        break;
}
```

DayOfYear Property

Sometimes, a program calls for the number of days passed since the start of the current year. The `DayOfYear` property handles this for us:

```
DateTime d1 = new DateTime(2013, 01, 05);  
Console.WriteLine(d1.DayOfYear); // 5  
  
d1 = new DateTime(2013, 08, 15);  
Console.WriteLine(d1.DayOfYear); // 277
```

Adding and Subtracting

We can use the TimeSpan structure to add or subtract date and time information from our values:

```
DateTime d1 = new DateTime(2013, 01, 01);  
  
// Timespans are days, hours, minutes, seconds  
TimeSpan s1 = new TimeSpan(20, 0, 0, 0);  
  
d1 += s1;  
Console.WriteLine(d1); // 1/21/2013  
  
d1 -= s1;  
Console.WriteLine(d1); // 1/1/2013
```

Add Methods

The DateTime struct also has some Add* methods which we can use without creating TimeSpans for simple cases:

```
DateTime theDate = new DateTime(2013, 12, 25);

theDate = theDate.AddYears(1);           // 12/25/2014 12:00:00 AM
theDate = theDate.AddMonths(2);          // 02/25/2015 12:00:00 AM
theDate = theDate.AddDays(1.5);          // 02/26/2015 12:00:00 PM
theDate = theDate.AddHours(-6);          // 02/26/2015 06:00:00 AM
theDate = theDate.AddMinutes(150);        // 02/26/2015 08:30:00 AM
theDate = theDate.AddSeconds(10.5);       // 02/26/2015 08:30:10 AM
theDate = theDate.AddMilliseconds(499);   // 02/26/2015 08:30:10.999 AM
theDate = theDate.AddTicks(10000);        // 02/26/2015 08:30:11 AM
```

Difference Between Dates

The DateTime subtract method will return a timespan telling us how far apart two dates are:

```
DateTime xmas = new DateTime(2013, 12, 25);  
DateTime independenceDay = new DateTime(2013, 7, 4);  
  
TimeSpan diff = xmas.Subtract(independenceDay);  
  
Console.WriteLine("There are {0} days from independence day " +  
    "until xmas", diff.Days);
```


DateTime Format Codes

The ToString() method on DateTime allows us to pass format codes for how we would like a date to be displayed.

<http://msdn.microsoft.com/en-us/library/zdtaw1bw.aspx>

There are built-in codes as well as custom formatters if you want to print it in a specific way.

```
DateTime xmas = new DateTime(2013, 12, 25);
Console.WriteLine(xmas.ToString("d"));
Console.WriteLine(xmas.ToString("MMM dd, yyyy"));
```

```
//      d: 6/15/2008
//      D: Sunday, June 15, 2008
//      f: Sunday, June 15, 2008 9:15 PM
//      F: Sunday, June 15, 2008 9:15:07 PM
//      g: 6/15/2008 9:15 PM
//      G: 6/15/2008 9:15:07 PM
//      m: June 15
//      o: 2008-06-15T21:15:07.0000000
//      R: Sun, 15 Jun 2008 21:15:07 GMT
//      s: 2008-06-15T21:15:07
//      t: 9:15 PM
//      T: 9:15:07 PM
//      u: 2008-06-15 21:15:07Z
//      U: Monday, June 16, 2008 4:15:07 AM
//      y: June, 2008
//
//      'h:mm:ss.ff t': 9:15:07.00 P
//      'd MMM yyyy': 15 Jun 2008
//      'HH:mm:ss.f': 21:15:07.0
//      'dd MMM HH:mm:ss': 15 Jun 21:15:07
//      '\Mon\t\h\: M': Month: 6
//      'HH:mm:ss.ffffzzz': 21:15:07.0000-07:00
```

Write a program to take a date and a number from the user, and print out the next number of Wednesdays.

LAB EXERCISE

Conclusion

Working with date and time information is common in developing applications.

Luckily, C# gives us a great class that not only stores dates, but allows us to easily perform common tasks like comparisons, additions, subtractions, and custom formatting.