

# DateTime vs DateTimeOffset

Asked 14 years ago   Modified 2 months ago   Viewed 462k times



What is the difference between a `DateTime` and a `DateTimeOffset` and when should one be used?

1147



Currently, we have a standard way of dealing with .NET `DateTime`s in a TimeZone-aware way: Whenever we produce a `DateTime` we do it in UTC (e.g. using `DateTime.UtcNow`), and whenever we display one, we convert back from UTC to the user's local time.



That works fine, but I've been reading about `DateTimeOffset` and how it captures the local and UTC time in the object itself.

`c#`

`.net`

`datetime`

`timezone`

`datetimeoffset`

Share

Improve this question

Follow

edited Feb 21, 2022 at 15:28



TylerH


21.2k ● 76 ● 79 ● 110

asked Dec 2, 2010 at 2:39



David Reis

13k ● 7 ● 39 ● 45

- 
- 1 When it comes to storage, [stackoverflow.com/questions/4715620/...](https://stackoverflow.com/questions/4715620/...) is interesting too.  
– Dejan Jul 21, 2016 at 18:22
- 
- 5 Curious people might also want to read [storing utc is not a silver bullet](#) – Jim Aho Feb 23, 2021 at 8:20 
- 

10 Answers

Sorted by:

Highest score (default)



1774



`DateTimeOffset` is a representation of *instantaneous time* (also known as *absolute time*). By that, I mean a moment in time that is universal for everyone (not accounting for [leap seconds](#), or the relativistic effects of [time dilation](#)). Another way to represent instantaneous time is with a `DateTime` where `.Kind` is `DateTimeKind.Utc`.

This is distinct from *calendar time* (also known as *civil time*), which is a position on someone's calendar, and there are many different calendars all over the globe. We call these calendars *time zones*. Calendar time is represented by a `DateTime` where `.Kind` is `DateTimeKind.Unspecified`, or `DateTimeKind.Local`. And `.Local` is only meaningful in scenarios where you have an implied understanding of where the computer that is using the result is positioned. (For example, a user's workstation)

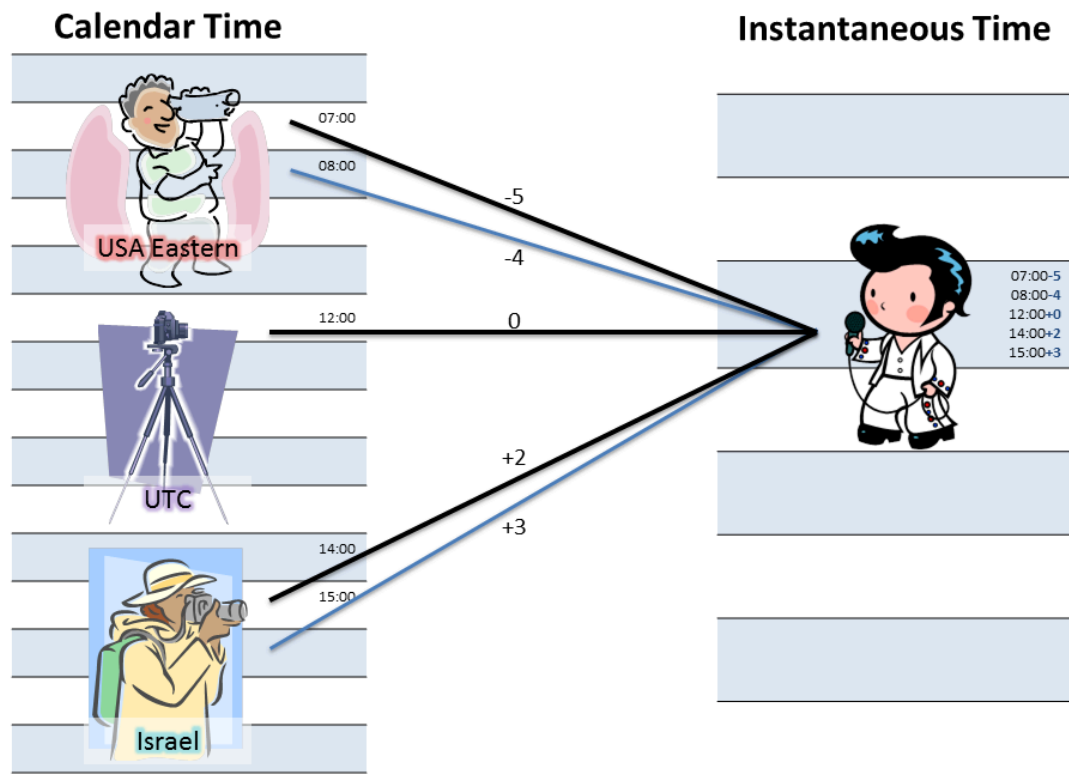
So then, why `DateTimeOffset` instead of a UTC `DateTime`? **It's all about perspective.** Let's use an analogy - we'll pretend to be photographers.

Imagine you are standing on a calendar timeline, pointing a camera at a person on the instantaneous timeline laid out in front of you. You line up your camera according to the rules of your timezone - which change periodically due to daylight saving time, or due to other changes to the legal definition of your time zone. (You don't have a steady hand, so your camera is shaky.)

The person standing in the photo would see the angle at which your camera came from. If others were taking pictures, they could be from different angles. This is what the `offset` part of the `DateTimeOffset` represents.

So if you label your camera "Eastern Time", sometimes you are pointing from -5, and sometimes you are pointing from -4. There are cameras all over the world, all labeled different things, and all pointing at the same instantaneous timeline from different angles. Some of them are right next to (or on top of) each other, so just knowing the offset isn't enough to determine which timezone the time is related to.

And what about UTC? Well, it's the one camera out there that is guaranteed to have a steady hand. It's on a tripod, firmly anchored into the ground. It's not going anywhere. We call its angle of perspective the zero offset.



So - what does this analogy tell us? It provides some intuitive guidelines-

- If you are representing time relative to some place in particular, represent it in calendar time with a `DateTime`. Just be sure you don't ever confuse one calendar with another. `Unspecified` should be your assumption. `Local` is only useful coming from `DateTime.Now`. For example, I might get `DateTime.Now` and save it in a database - but when I retrieve it, I have to assume that it is `Unspecified`. I can't rely that my local calendar is the same calendar that it was originally taken from.
- If you must always be certain of the moment, make sure you are representing instantaneous time. Use `DateTimeOffset` to enforce it, or use UTC `DateTime` by convention.

- If you need to track a moment of instantaneous time, but you want to also know "What time did the user think it was on their local calendar?" - then you *must* use a `DateTimeOffset`. This is very important for timekeeping systems, for example - both for technical and legal concerns.
- If you ever need to modify a previously recorded `DateTimeOffset` - you don't have enough information in the offset alone to ensure that the new offset is still relevant for the user. You must *also* store a timezone identifier (think - I need the name of that camera so I can take a new picture even if the position has changed).

It should also be pointed out that [Noda Time](#) has a representation called `ZonedDateTime` for this, while the .Net base class library does not have anything similar. You would need to store both a `DateTimeOffset` and a `TimeZoneInfo.Id` value.

- Occasionally, you will want to represent a calendar time that is local to "whomever is looking at it". For example, when defining what *today* means. Today is always midnight to midnight, but these represent a near-infinite number of overlapping ranges on the instantaneous timeline. (In practice we have a finite number of timezones, but you can express offsets down to the tick) So in these situations, make sure you understand how to either limit the "who's asking?" question down to a single time zone, or deal with translating them back to instantaneous time as appropriate.

Here are a few other little bits about `DateTimeOffset` that back up this analogy, and some tips for keeping it straight:

- If you compare two `DateTimeOffset` values, they are first normalized to zero offset before comparing. In other words, `2012-01-01T00:00:00+00:00` and `2012-01-01T02:00:00+02:00` refer to the same instantaneous moment, and are therefore equivalent.
- If you are doing any unit testing and need to be certain of the offset, test *both* the `DateTimeOffset` value, and the `.offset` property separately.
- There is a one-way implicit conversion built in to the .Net framework that lets you pass a `DateTime` into any `DateTimeOffset` parameter or variable. When doing so, **the `.Kind` matters**. If you pass a UTC kind, it will carry in with a zero offset, but if you pass either `.Local` or `.Unspecified`, it will assume to be **local**. The framework is basically saying, "Well, you asked me to convert calendar time to instantaneous time, but I have no idea where this came from, so I'm just going to use the local calendar." This is a huge gotcha if you load up an unspecified `DateTime` on a computer with a different timezone. (IMHO - that should throw an exception - but it doesn't.)

### Shameless Plug:

Many people have shared with me that they find this analogy extremely valuable, so I included it in my Pluralsight course, [Date and Time Fundamentals](#). You'll

find a step-by-step walkthrough of the camera analogy in the second module, "Context Matters", in the clip titled "Calendar Time vs. Instantaneous Time".

Share Improve this answer

edited Mar 3, 2020 at 18:00

Follow



John Smith

7,399 ● 7 ● 51 ● 63

answered Jan 10, 2013 at 22:09



Matt Johnson-Pint

241k ● 75 ● 462 ● 607

---

12 @ZackJannsen If you have a `DateTimeOffset` in C#, then you should persist that to a `DATETIMEOFFSET` in SQL Server. `DATETIME2` or just `DATETIME` (depending on the range required) are fine for regular `DateTime` values. Yes - you can resolve a local time from any pairing of timezone + dto or utc. The difference is - do you always want to be computing the rules with each resolve, or do you want to precalculate them? In many cases (sometimes for legal concerns) a DTO is a better choice. – Matt Johnson-Pint Mar 29, 2013 at 4:15

---

4 @ZackJannsen For the second part of your question, I would recommend doing as much as possible server-side. Javascript is not that great for timezone calculation. If you must do it, use one of [these libraries](#). But server side is best. If you have other more detailed questions, please start a new S.O. question for them and I will answer if I can. Thanks. – Matt Johnson-Pint Mar 29, 2013 at 4:20


---

5 @JoaoLeme - That depends on where you obtained it from. You are correct that if you say `DateTimeOffset.Now` on the server, you will indeed get the server's offset. The point is that the `DateTimeOffset` type can retain that offset. You could just as easily do that on the client, send it to the

server, and then your server would know the client's offset.

– [Matt Johnson-Pint](#) Aug 9, 2013 at 19:21 

6 Yes, that's correct. Except that DTO is stored as the (local time, offset) pair, not the (utc time, offset) pair. In other words, the offset from UTC is already reflected in the local time. To convert back to utc, invert the sign of the offset and apply it to the local time. – [Matt Johnson-Pint](#) Feb 27, 2014 at 20:15

4 I think timezone and utc have very little to do with Cameras and the angle of a photographer. Take your kids travelling across timezones and even a 7 year old can understand it. – [hamish](#) Aug 24, 2014 at 21:52 



From Microsoft:

**501**



These uses for DateTimeOffset values are much more common than those for DateTime values. As a result, DateTimeOffset should be considered the default date and time type for application development.

source: ["Choosing Between DateTime, DateTimeOffset, TimeSpan, and TimeZoneInfo"](#), MSDN

We use `DateTimeOffset` for nearly everything as our application deals with particular points in time (e.g. when a record was created/updated). As a side note, we use `DATETIMEOFFSET` in SQL Server 2008 as well.

I see `DateTime` as being useful when you want to deal with dates only, times only, or deal with either in a generic



sense. For example, if you have an alarm that you want to go off every day at 7 am, you could store that in a `DateTime` utilizing a `DateTimeKind` of `Unspecified` because you want it to go off at 7am regardless of DST. But if you want to represent the history of alarm occurrences, you would use `DateTimeOffset`.

Use caution when using a mix of `DateTimeOffset` and `DateTime` especially when assigning and comparing between the types. Also, only compare `DateTime` instances that are the same `DateTimeKind` because `DateTime` ignores timezone offset when comparing.

Share Improve this answer

Follow

edited Aug 9, 2017 at 3:54



Cœur

38.6k ● 26 ● 202 ● 276

answered Jan 9, 2013 at 19:42




Clay

11.5k ● 5 ● 49 ● 46

---

16 I'll just say that I like this answer too, and upvoted. Though in the last part - even ensuring `Kind` are the same, comparison could be in error. If both sides have `DateTimeKind.Unspecified` you don't really know that they came from the same time zone. If both sides are `DateTimeKind.Local`, *most* comparisons are going to be fine, but you could still have errors if one side is ambiguous in the local time zone. Really only `DateTimeKind.Utc` comparisons are foolproof, and yes, `DateTimeOffset` is usually preferred. (Cheers!) – [Matt Johnson-Pint](#) Apr 22, 2016 at 20:25

---

- 4 +1 I'd add to this: The `DataType` you choose should reflect your intent. Do not use `DateTimeOffset` everywhere, just cause. If the Offset matters for your Calculations and Reading-From/Persisting-To the DataBase, then use `DateTimeOffset`. If it doesn't matter, then use `DateTime`, so you understand (just by looking at the `DataType`) that the Offset should have no bearing and Times should remain relative to the Locality of the Server/Machine your C# Code is running on. – [MikeTeeVee](#) Jul 22, 2019 at 18:50 
- 



`DateTime` is capable of storing only two distinct times, the local time and UTC. The *Kind* property indicates which.

143



`DateTimeOffset` expands on this by being able to store local times from anywhere in the world. It also stores the *offset* between that local time and UTC. Note how `DateTime` cannot do this unless you'd add an extra member to your class to store that UTC offset. Or only ever work with UTC. Which in itself is a fine idea btw.



Share Improve this answer

answered Dec 2, 2010 at 17:47

Follow



[Hans Passant](#)

940k ● 148 ● 1.7k ● 2.6k

- 
- 9 by far the easiest ans simplest answer of this post. – [Yann](#) Mar 31, 2023 at 11:17
- 



`DateTime.Now`

Fri 03 Dec 21 18:40:11

85



`DateTimeOffset.Now`

Fri 03 Dec 21 18:40:11 +02:00

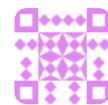


So, `DateTimeOffset` stores information about how the time relates to UTC, basically the time zone.

Share Improve this answer

answered Dec 3, 2021 at 17:41

Follow



**Tessaract**

1,885 ● 1 ● 14 ● 30

---

10 Underrated answer, this should be on top – [DexterHaxxor](#)  
Aug 8, 2022 at 14:33

---

5 I suggest refining "basically the time zone" --> "basically the offset component of the time zone". See remark in the docs about why an offset is not equivalent to a time zone:  
[learn.microsoft.com/en-us/dotnet/api/...](https://learn.microsoft.com/en-us/dotnet/api/...) – [tsemer](#) Jan 5 at 8:07

---



62



This piece of code from [Microsoft](#) explains everything:

```
// Find difference between Date.Now and Date.UtcNow
date1 = DateTime.Now;
date2 = DateTime.UtcNow;
difference = date1 - date2;
Console.WriteLine("{0} - {1} = {2}", date1, date2, difference);

// Find difference between Now and UtcNow using DateTimeOffset
dateOffset1 = DateTimeOffset.Now;
dateOffset2 = DateTimeOffset.UtcNow;
difference = dateOffset1 - dateOffset2;
Console.WriteLine("{0} - {1} = {2}", dateOffset1, dateOffset2, difference);

// If run in the Pacific Standard time zone on 4/2/2021
// displays the following output to the console:
```

```
// 4/2/2007 7:23:57 PM - 4/3/2007 2:23:57 AM = -0
// 4/2/2007 7:23:57 PM -07:00 - 4/3/2007 2:23:57
```

Share Improve this answer

answered Oct 19, 2018 at 9:32

Follow



[Mojtaba](#)

3,492 ● 1 ● 23 ● 26

---

So lets say I need to store a CreatedDate property when the user created something. Do I pass DateTimeOffset.Now or.UtcNow to the server? – [Morten\\_564834](#) Mar 29, 2021 at 21:23

---

@Morten\_564834 I would say `DateTimeOffset.Now` because you can then compare `CreatedDate` irrespective of their timezone. – [Mojtaba](#) Mar 30, 2021 at 7:34

---

@Morten\_564834 @Mojtaba using `DateTimeOffset.Now` tells you also in which timezone it was created. But personally I would use `DateTime.UtcNow` for these technical fields. (But yes that requires people to use utc) which you could check for in a property accesor. – [Wouter](#) Sep 27, 2022 at 15:23

---

1 @Wouter just to be more accurate, `DateTimeOffset.Now` doesn't tell you which timezone it was, just what offset it was from. A timezone can have multiple offsets and same offset can be shared among many timezones. But I get what you are saying. – [nawfal](#) Feb 17, 2023 at 11:28

---

1 @Wouter FYI I can see the examples in the link from nawfal, e.g.: `+01:00` could belong to the following time zones: `(GMT+01:00) Amsterdam (GMT+01:00) Belgrade (GMT+01:00) Brussels (GMT+01:00) Sarajevo (GMT+01:00) West Central Africa` – [Stajs](#) Mar 9, 2023 at 20:20

---



The most important distinction is that `DateTime` does not store time zone information, while `DateTimeOffset` does.

44



Although `DateTime` distinguishes between UTC and Local, there is absolutely no explicit time zone offset associated with it. If you do any kind of serialization or conversion, the server's time zone is going to be used.



Even if you manually create a local time by adding minutes to offset a UTC time, you can still get bit in the serialization step, because (due to lack of any explicit offset in `DateTime`) it will use the server's time zone offset.

For example, if you serialize a `DateTime` value with `Kind=Local` using `Json.Net` and an ISO date format, you'll get a string like `2015-08-05T07:00:00-04`. Notice that last part (-04) had nothing to do with your `DateTime` or any offset you used to calculate it... it's just purely the server's time zone offset.

Meanwhile, `DateTimeOffset` explicitly includes the offset. It may not include the name of the time zone, but at least it includes the offset, and if you serialize it, you're going to get the explicitly included offset in your value instead of whatever the server's local time happens to be.

Share Improve this answer

answered Aug 5, 2015 at 23:00

Follow



Triynko

19.2k ● 21 ● 112 ● 181

---

30 with all the above answers, I wonder why no one bothered to write your single sentence that sums it all up `The most important distinction is that DateTime does not store time zone information, while DateTimeOffset does.` – [Korayem](#) May 15, 2016 at 22:38


---

17 `DateTimeOffset` does NOT store time zone info. MS doc titled "Choosing between `DateTime`, `DateTimeOffset`, `TimeSpan`, and `TimeZoneInfo`" specifies this stating: "A `DateTimeOffset` value is not tied to a particular time zone, but can originate from any of a variety of time zones". That said, `DateTimeOffset` IS time zone AWARE, containing the offset from UTC, which makes all the difference and is why it's MS recommended default class when dealing with app development that deals with date info. If you truly care about which specific timezone the data came from, you must preserve that separately – [stonedauwg](#) Jun 14, 2017 at 20:35

---

2 Yes, but as has been shown in many places, + or - hours says nothing about what timezone you were in and is ultimately useless. Depending on what you need to do, you can just as well store a datetime as `Kind.Unspecified` and then store the id of its timezone and I think you are actually better off. – [Arwin](#) Apr 6, 2018 at 13:14

---

1 It's true that an offset isn't the same thing as time zone, but it's far simpler to understand and is usually what is important, far from useless. It is true that a timestamp with an offset of -5 could have happened in Eastern Standard Time or in Central Daylight Time. But if I don't care what time zone it was in, if I just want to know when the event happened (which is what a timestamp is usually for), the offset is more useful, because I don't need to consult a historical, time zone database in order to convert it to UTC. – [Corrodias](#) May 16, 2023 at 17:07 

---



41



**TLDR** if you don't want to read all these great answers :-)

### Explicit:

Using `DateTimeOffset` because the timezone is forced to UTC+0.

### Implicit:

Using `DateTime` where you *hope* everyone sticks to the unwritten rule of the timezone always being UTC+0.

---

(Side note for devs: **explicit is always better than implicit!**)

(Side side note for Java devs, C# `DateTimeOffset` == Java `OffsetDateTime`, read this:

<https://www.baeldung.com/java-zoneddatetime-offsetdatetime>)

Share Improve this answer

edited Sep 17, 2020 at 12:03

Follow

answered Aug 18, 2020 at 13:28



**Blundell**

76.3k ● 31 ● 217 ● 239

- 
- 2 If you're running in Azure you don't have to worry that everyone sticks to the unwritten rule. `DateTime.Now`, `DateTimeOffset.Now`, `DateTime.UtcNow` and `DateTimeOffset.UtcNow` all return the exact same point in time in UTC. – [Craig W.](#) Sep 29, 2020 at 16:29
- 



36



There's a few places where `DateTimeOffset` makes sense. One is when you're dealing with recurring events and daylight savings time. Let's say I want to set an alarm to go off at 9am every day. If I use the "store as UTC, display as local time" rule, then the alarm will be going off at a *different* time when daylight savings time is in effect.

There are probably others, but the above example is actually one that I've run into in the past (this was before the addition of `DateTimeOffset` to the BCL - my solution at the time was to explicitly store the time in the local timezone, and save the timezone information along side it: basically what `DateTimeOffset` does internally).

Share Improve this answer

answered Dec 2, 2010 at 2:59

Follow



[Dean Harding](#)

72.6k ● 13 ● 148 ● 180

- 
- 18 `DateTimeOffset` doesn't fix the DST problem – [JarrettV](#) Jan 15, 2013 at 18:47
-



4 Using TimeZoneInfo class does carry rules for DST. if you are on .net 3.5 or later then use either TimeZone or TimeZoneInfo classes to deal with dates that must handle Daylight Savings Time in conjunction with the timezone offset. – [Zack Jannsen](#) Mar 29, 2013 at 11:41

---

1 Yes good example of an exception (the alarm app) but when the time is more important than the date you should really store that separate in your schedule data structure for the application, i.e. occurrence type = Daily and time = 09:00. The point here is the developer needs to be aware of what type of date they are recording, calculating or presenting to users. Especially apps tend to be more global now we have the internet as standard and big app stores to write software for. As a side note I'd also like to see Microsoft add a separate Date and Time structure. – [Tony Wall](#) Apr 9, 2014 at 9:12

---

5 Summarizing Jarrett's and Zack's comments: It sounds like DateTimeOffset *alone* will not handle the DST problem but using DateTimeOffset in conjunction with TimeZoneInfo will handle it. This is no different from DateTime where kind is Utc. In both cases I must know the time zone (not just the offset) of the calendar I am projecting the moment to. (I might store that in a user's profile or get it from the client (e.g. Windows) if possible). Sound right? – [Jeremy Cook](#) Nov 8, 2014 at 5:31

---

3 "There's a few places where DateTimeOffset makes sense."  
--- Arguably, it more often makes sense than not.  
– [Ronnie Overby](#) Dec 5, 2018 at 20:28

---



A major difference is that `DateTimeOffset` can be used in conjunction with `TimeZoneInfo` to convert to local times in timezones other than the current one.



This is useful on a server application (e.g. ASP.NET) that is accessed by users in different timezones.



Share Improve this answer

answered Dec 2, 2010 at 13:03

Follow



to StackOverflow

125k ● 33 ● 210 ● 341

3 @Bugeo Bugeo is true, but there is a risk. You can compare two `DateTime`s by first calling "`ToUniversalTime`" on each. If you have exactly one value in the comparison that is `DateTimeKind = Unspecified` your strategy will fail. This potential for a failure is a reason to consider `DateTimeOffset` over `DateTime` when conversions to local time are required.

– Zack Janssen Mar 29, 2013 at 11:49

As above, I think in this scenario you are better off with storing the `TimeZoneId` than using `DateTimeOffset`, which ultimately means nothing. – Arwin Apr 6, 2018 at 13:15

Or you could store a `DateTimeOffset` plus the `TimeZoneId`. Then you'd not only have the offset but also the time zone that caused the offset. Keep in mind that multiple time zones can share the same offset. – Manfred Jul 25, 2020 at 7:00



5

The only negative side of `DateTimeOffset` I see is that Microsoft "forgot" (by design) to support it in their `XmlSerializer` class. But it has since been added to the `XmlConvert` utility class.



[XmlConvert.ToDateTimeOffset](#)



[XmlConvert.ToString](#)



I say go ahead and use DateTimeOffset and TimeZoneInfo because of all the benefits, just beware when creating entities which will or may be serialized to or from XML (all business objects then).

Share Improve this answer

edited Aug 27, 2018 at 19:41

Follow

answered Apr 9, 2014 at 11:11



Tony Wall

1,412 ● 22 ● 18

---