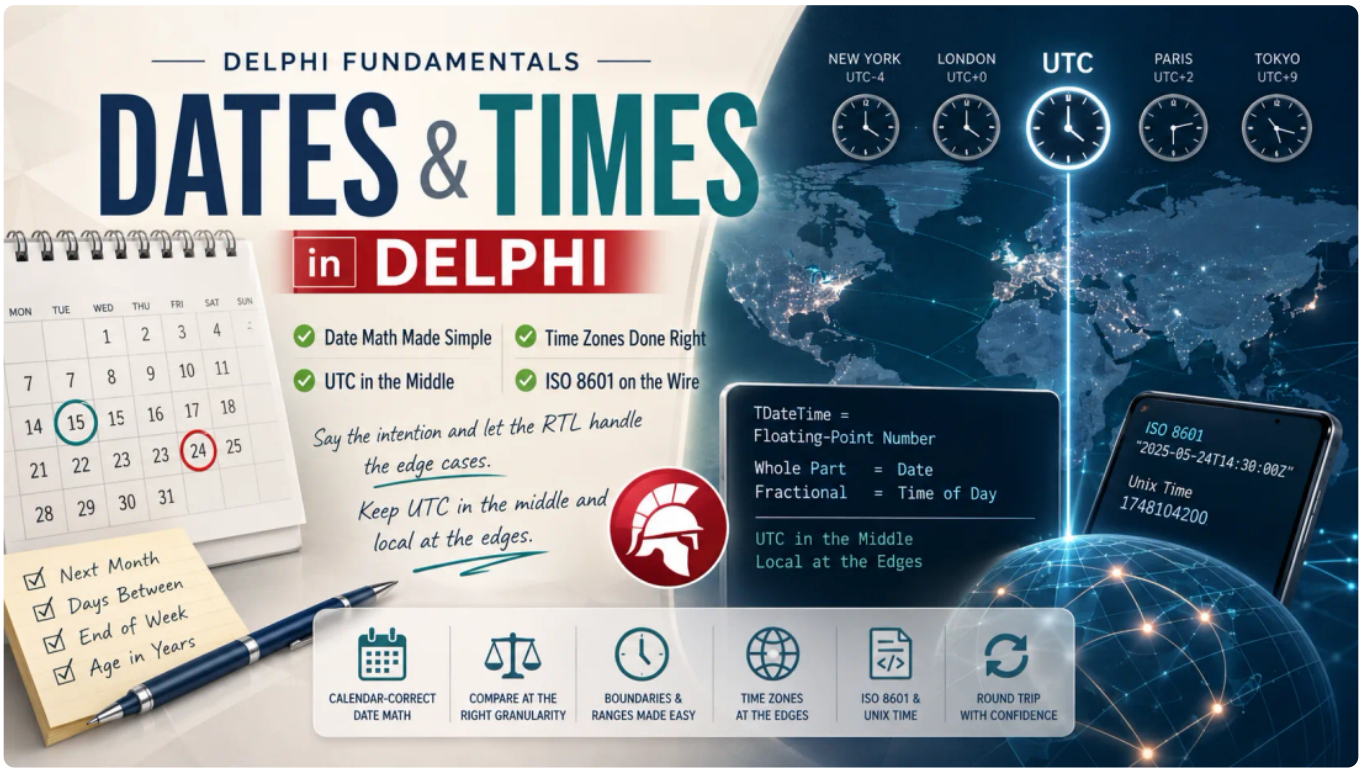
TUTORIAL · AUGUST 3, 2026

# Datum und Uhrzeit in Delphi

Ein zweiteiliger Kurs zum Umgang mit Datum und Uhrzeit für Delphi-Entwickler — von dem, was ein TDateTime wirklich ist, und den DateUtils-Einzeilern, die handgeschriebene Kalenderarithmetik ersetzen, bis zur goldenen Regel für Zeitzonen, den Umrechnungen mit TTimeZone und den Austauschformaten, die Zeitstempel zwischen Systemen eindeutig halten.

By Dr. Holger Flick




Datumscode hat die Eigenschaft, auch erfahrene Entwickler still und leise zu demütigen. Sie brauchen „den ersten Tag des nächsten Monats", oder „wie viele Tage bis zum Stichtag", oder das Alter einer Person in Jahren — und schon zerlegen Sie ein Datum in seine Bestandteile, rechnen darauf herum, überlegen, wie viele Tage der Februar dieses Jahr hat, und setzen das Ergebnis wieder zusammen. Es funktioniert. Es braucht aber auch zwanzig Zeilen für etwas, das eine einzige verdient hätte.

Und Monate später geht die Anwendung an einen Kunden drei Zeitzonen weiter, und eine zweite, unangenehmere Fehlerklasse taucht auf: Termine, die eine Stunde daneben liegen, Berichte mit den Daten von gestern, ein „erstellt am"-Zeitstempel, der irgendwie in der Zukunft liegt. Jeder, der Software über Zeitzonen hinweg ausgeliefert hat, kennt dieses Gespenst.

Dieses Tutorial ist der Kurs für beide Hälften des Problems, ausgeliefert als zweiteilige Artikelserie und vollständig auf der RTL-Unit `System.DateUtils` aufgebaut — einer Unit, die seit Delphi 6 mitgeliefert wird und die überraschend viele Entwickler nie geöffnet haben. Teil 1 macht die alltägliche Datumsrechnung lesbar und kalendarisch korrekt. Teil 2 nimmt die Annahme, auf die sich Teil 1 stillschweigend stützt — dass alle Ihre Datumswerte in derselben Zeitzone liegen — und räumt sie ab.

Lesen Sie die beiden Teile der Reihe nach. Der zweite setzt genau dort an, wo der erste aufhört, und seine ganze Disziplin ergibt sich erst, wenn Sie wissen, was ein `TDateTime` eigentlich ist.

## Teil 1 — Alltägliche Datumsrechnung, ohne Umstände

DateUtils, Teil 1: Datum und Uhrzeit in Delphi

Es beginnt mit einer Tatsache, die erstaunlich viel Datumsverwirrung auflöst: Ein `TDateTime` ist eine einzige Fließkommazahl, und sobald Sie sehen, was ihr ganzzahliger und ihr gebrochener Teil bedeuten, werden sowohl die klassischen Rechentricks *als auch* der Grund, warum sie bei Monaten und Jahren scheitern, unmittelbar einleuchtend. Von dort geht es im gewohnten Vergleich „früher gegen heute" durch die alltäglichen Operationen: Kalendereinheiten addieren und subtrahieren, Zeitspannen zwischen zwei Datumswerten messen (das Alter in einer lesbaren Zeile), Datumswerte in der Granularität vergleichen, die Sie tatsächlich meinen, statt bis auf die Millisekunde, und auf Grenzen einrasten wie „Anfang dieser Woche" oder „der letzte Moment dieses Monats", Schaltjahre inklusive. In den `XxxBetween`-Funktionen steckt eine semantische Falle, in die beim ersten Mal fast jeder tappt — und zu wissen, welche Frage Sie eigentlich stellen, ist der ganze Trick fehlerfreier Datumsrechnung. Der Beitrag ist auch ehrlich über die Grenzen: Er benennt die Fälle, in denen die klassischen Funktionen weiterhin die richtige Wahl sind.

## Teil 2 — Zeitzonen, UTC und die richtige Form auf der Leitung

DateUtils, Teil 2: Zeitzonen, UTC und ISO 8601

Hier geht es nicht mehr um Lesbarkeit, sondern um Korrektheit. Der Beitrag beginnt mit der einen Regel, auf die sich die ganze Branche geeinigt hat — der Disziplin, die die meisten Zeitzonen-Katastrophen verhindert, noch bevor irgendeine API ins Spiel kommt — und zeigt dann die RTL-Werkzeuge, die sie umsetzen: den aktuellen Moment in UTC festhalten und an den Rändern mit der Klasse `TTimeZone` umrechnen. Unterwegs erklärt er, warum jede dieser Umrechnungen ein *konkretes Datum* entgegennimmt und warum die Abkürzung, die gleichwertig aussieht — einen festen Offset selbst zu addieren —, ungefähr das halbe Jahr über stillschweigend falsch ist. Die zweite Hälfte behandelt die beiden Formate, die einen Zeitstempel eindeutig halten, sobald er Ihr Programm verlässt, ISO 8601 und Unix-Zeit, samt dem kleinen Parameter dieser Funktionen, über den die meisten stolpern. Zum Schluss steht der komplette Rundlauf in wenigen Zeilen — und eine ehrliche Einordnung der zwei Szenarien, in denen die RTL allein nicht ausreicht.

---

## Was Sie danach können

Arbeiten Sie beide Teile durch, und der Umgang mit Datum und Uhrzeit hört auf, der Teil der Codebasis zu sein, dem Sie sich vorsichtig nähern:

- **Erklären, was ein `TDateTime` ist** — und damit sowohl die Rechentricks als auch ihre Grenzen.
- **Kalendarische Absichten direkt ausdrücken** — „nächster Monat", „Tage dazwischen", „Ende der Woche" — statt sie selbst zu bauen.
- **Datumswerte in der gemeinten Granularität vergleichen** und erkennen, wann „between" etwas anderes zählt, als Sie angenommen haben.
- **Beliebige Zeiträume abgrenzen** — Tag, Woche, Monat, Jahr — ohne Tage zu zählen oder Schaltjahre gesondert zu behandeln.
- **Zeitangaben über Zonen hinweg korrekt speichern und übertragen**, nur an den Rändern umrechnen, immer datumsabhängig.

- **Zeitstempel so auf die Leitung geben**, dass das empfangende System sie nicht missverstehen kann — und wissen, wann Sie über die RTL hinausgreifen müssen.

> Sagen Sie die Absicht und überlassen Sie der RTL die Sonderfälle; halten Sie UTC in der Mitte und die lokale Zeit an den Rändern. Diese beiden Gewohnheiten decken fast jeden Datumsfehler ab, den Sie sonst ausliefern würden.

INDEPENDENT · AD-FREE · READER-SUPPORTED

# Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

## Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

## Ways to support

KO-FI

### Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

### Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

### Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

**Support on Ko-fi —**