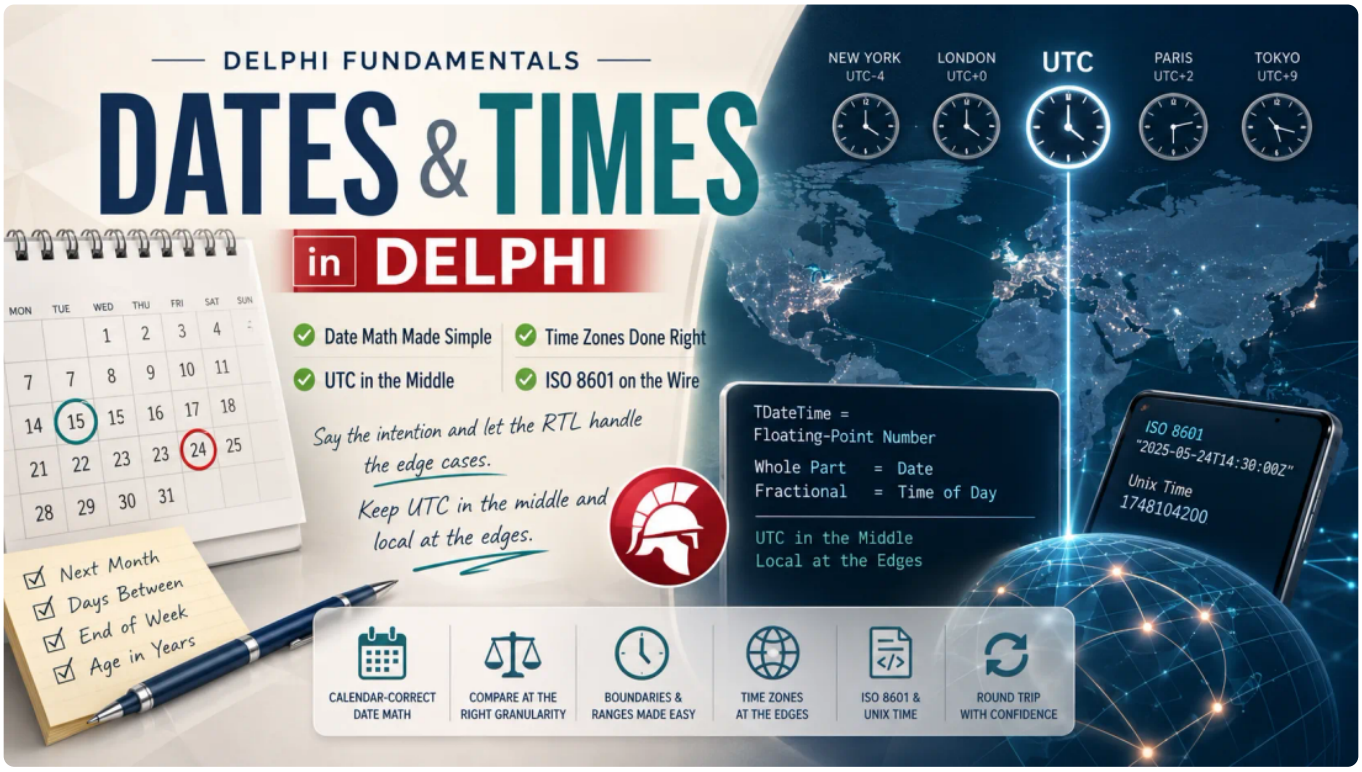
TUTORIAL · AUGUST 3, 2026

Dates and Times in Delphi

A two-part course on date and time handling for Delphi developers — from what a TDateTime really is and the DateUtils one-liners that replace hand-rolled calendar math, to the golden rule of time zones, TTimeZone conversions, and the interchange formats that keep timestamps unambiguous between systems.

By Dr. Holger Flick



Date code has a way of quietly humbling experienced developers. You need "the first day of next month," or "how many days until the deadline," or someone's age in years — and you find yourself decoding a date into pieces, doing arithmetic on them, worrying about how many days February has this year, and reassembling the result. It works. It also takes twenty lines to say something that should take one.

Then, months later, the app ships to a customer three time zones away and a second, nastier category of bug arrives: appointments an hour off, reports showing yesterday's data, a "created at" timestamp somehow in the future. Everyone who has built software that crosses time zones has met that ghost.

This tutorial is the course for both halves of the problem, delivered as a two-part article series built entirely on the RTL's `System.DateUtils` — a unit that has shipped with Delphi since version 6 and that a surprising number of developers have never opened. Part 1 makes everyday date math readable and calendar-correct. Part 2 takes the assumption Part 1 quietly relies on — that all your dates live in the same zone — and removes it.

Read the two parts in order. The second one starts exactly where the first ends, and its whole discipline only makes sense once you know what a `TDateTime` actually is.

Part 1 — Everyday date math, without the ceremony

DateUtils, Part 1: Modern Date and Time in Delphi

It starts with one fact that dissolves a startling amount of date confusion: a `TDateTime` is a single floating-point number, and once you see what its whole and fractional parts mean, both the classic arithmetic tricks *and* the reason they break on months and years become obvious. From there it's a paired old-way-vs-new-way tour of the everyday operations — adding and subtracting calendar units, measuring spans between two dates (age in one readable line), comparing dates at the granularity you actually mean rather than down to the millisecond, and snapping to boundaries like "start of this week" or "the last moment of this month," leap years included. There's one semantic trap in the `XxxBetween` functions that catches nearly everyone the first time, and knowing which question you're really asking turns out to be the whole trick to bug-free date math. The post is honest about scope, too: it names the cases where the classic functions are still the right call.

Part 2 — Time zones, UTC, and getting it right on the wire

DateUtils, Part 2: Time Zones, UTC, and ISO 8601

This is where date handling stops being about readability and starts being about correctness. It opens with the one rule the whole industry has converged on — the discipline that prevents most time-zone disasters before any API is involved — and then shows the RTL tools that implement it: stamping the current moment in UTC, and converting at the edges with the `TTimeZone` class. Along the way it explains why every one of those conversions takes a *specific date*, and why the shortcut that looks equivalent — adding a fixed offset yourself — is silently wrong for roughly half the year. The second half covers the two formats that keep a timestamp unambiguous once it leaves your program, ISO 8601 and Unix time, including the small parameter on those functions that people most often trip over. It closes with the complete round trip in a handful of lines, and an honest account of the two scenarios where the RTL alone isn't enough.

What you'll be able to do afterwards

Work through both parts and date handling stops being the part of the codebase you approach carefully:

- **Explain what a `TDateTime` is** — and why that explains both the arithmetic tricks and their limits.
- **Express calendar intentions directly** — "next month," "days between," "end of the week" — instead of hand-rolling them.
- **Compare dates at the granularity you mean**, and know when "between" is counting something different from what you assumed.
- **Bracket any period** — day, week, month, year — without counting days or special-casing leap years.
- **Store and transmit time correctly across zones**, converting only at the edges, always date-aware.
- **Put timestamps on the wire** in a format the receiving system can't misread — and know when to reach beyond the RTL.

> Say the intention and let the RTL handle the edge cases; keep UTC in the middle and local at the edges. Those two habits cover almost every date bug you'd otherwise ship.

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 —