TUTORIAL · AUGUST 15, 2026

Holiday Calendars in Delphi

A two-part course on computing national public holidays in Delphi — turning holiday rules into data instead of a case statement, the three date primitives the entire US federal calendar is built from, the Gauss/Meeus Easter algorithm in fifteen lines, and merging two countries into one calendar that never needs updating.

By Dr. Holger Flick

Every business application eventually has to answer a deceptively simple question: *is this day a working day?*

Payroll asks it. Delivery estimates ask it. Anything that promises a response "within five business days" asks it. And the first implementation, in almost every codebase, is a `case` statement that grows a new branch every December — because someone hard-coded the dates for next year and the year after that ran out.

The fix isn't a bigger `case` statement. It's noticing that a holiday isn't a date at all — it's a *rule*. Christmas is always 25 December. Thanksgiving is the fourth Thursday in November. Memorial Day is the *last* Monday in May, which is not the same as the fourth or the fifth. And Ostermontag is whatever day follows a lunisolar calculation standardized fifteen centuries before the first computer. Encode the dates and you're back next year. Encode the rules and you're done forever.

This tutorial is that engine, delivered as a two-part article series and built entirely on the RTL unit `System.DateUtils` — no third-party dependencies, roughly a hundred and twenty lines in total. Part 1 builds the data structure and covers the United States. Part 2 takes on Easter, Germany, and the moment two national calendars have to live in the same table.

Read the parts in order — Part 2 extends Part 1's structure without changing a line of it, which is precisely the point it's making.

Part 1 — Rules as data, and the US federal calendar

Two Countries, Twenty Holidays, One Data Structure

It starts with one observation that collapses the whole problem: whatever a holiday means culturally, computationally it is a function that takes a year and returns a date. Give every rule that signature — a `reference to function(Year): TDate` in a small record, alongside a stable UID that survives display names being translated or amended by legislation — and the code that evaluates them has exactly one path through it, with no branching on rule type anywhere. From there it's three tiny primitives: a fixed date, the *n*th weekday of a month, and the last weekday of a month. Those three cover the entire US statute, and all eleven federal holidays fall out as a single readable array that reads almost like the law it implements. The post is also blunt about the bug that survives testing in January and detonates in September — the one where "last Monday in May" gets written as "fourth" or "fifth" and quietly hands you a date in June.

Part 2 — Easter, the German calendar, and merging two countries

Four German Holidays Hang Off One Number

Germany is where the design gets tested, because four of its nine nationwide holidays aren't anchored to the Gregorian calendar at all — they hang off Easter, which moves across a 35-day window and has no closed form you can look up. The good news: the anonymous Gregorian algorithm fits in fifteen lines of integer arithmetic, and once you have that one number, Karfreitag, Ostermontag, Christi Himmelfahrt, and Pfingstmontag are one `IncDay` call each — though two of those offsets are not the numbers tradition quotes, and the off-by-one is silent. Then come the parts nobody warns you about: Germany's nine holidays are an *overlap* of sixteen state laws rather than a federal list, the two countries handle weekend holidays in exactly opposite ways, and merging both calendars surfaces a date collision that only happens in certain years — which is enough to crash a naive index in production, in the one year it occurs. The evaluation loop from Part 1 absorbs all of it unchanged.

What you'll be able to do afterwards

- Model any holiday — fixed, ordinal weekday, terminal weekday, or Easter-relative — as one uniform `function(Year): TDate` and evaluate the whole calendar in a loop with no type branching.
- Generate the complete US federal calendar and Germany's nationwide calendar for any year, with `System.DateUtils` and nothing else.
- Compute Easter Sunday from first principles, and know why the resulting date is the Western one and what to reach for if you need the Orthodox date.
- Keep *occurrence* and *observation* as separate dates, so "is today Christmas?" and "is the office closed?" stay two answerable questions instead of one lossy field.
- Merge multiple countries into a single calendar, survive two holidays landing on the same day, and compute business-day arithmetic on top of it.
- Judge honestly when to stop: the post names the maintained open-source alternatives and the point at which reaching for one beats owning the code.

> Encode the rule, not the date. A holiday table built from rules is written once and is correct in every year you never tested.

Both parts build on the Dates and Times in Delphi tutorial — worth reading first if `TDateTime` has ever surprised you.

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 —