NEWS · AUGUST 14, 2026

Vier deutsche Feiertage hängen an einer einzigen Zahl

Teil zwei einer zweiteiligen Serie über die Berechnung gesetzlicher Feiertage in Delphi — der Gaußsche Osteralgorithmus in fünfzehn Zeilen, die vier daraus abgeleiteten deutschen Feiertage und das Zusammenführen zweier nationaler Kalender mit der Regeln-als-Daten-Struktur aus Teil 1.

By Dr. Holger Flick

Teil 1 hat eine Feiertags-Engine aus Daten statt aus Verzweigungen gebaut: ein Record mit einem stabilen

Bezeichner, einem Anzeigenamen und einer anonymen Methode, die ein Jahr in ein Datum verwandelt. Elf US-Bundesfeiertage passen mit drei kleinen Primitiven in diese Struktur — einem festen Datum, dem *n*-ten

Wochentag eines Monats und dem letzten Wochentag eines Monats — und der auswertende Code verzweigt überhaupt nicht nach Regeltyp.

Deutschland ist der Ort, an dem sich dieser Entwurf bewähren muss, denn der deutsche Kalender enthält eine Regelart, die der amerikanische nicht kennt. Vier der neun bundeseinheitlichen Feiertage sind gar nicht im gregorianischen Kalender verankert. Sie hängen an Ostern — und Ostern ist kein Datum, das man in einem Gesetz nachschlagen kann. Es ist das Ergebnis einer Berechnung, die einen Sonnenkalender mit einem Mondkalender in Einklang bringt, 325 n. Chr. auf dem Konzil von Nicäa vereinheitlicht, und es wandert von Jahr zu Jahr über ein Fenster von 35 Tagen.

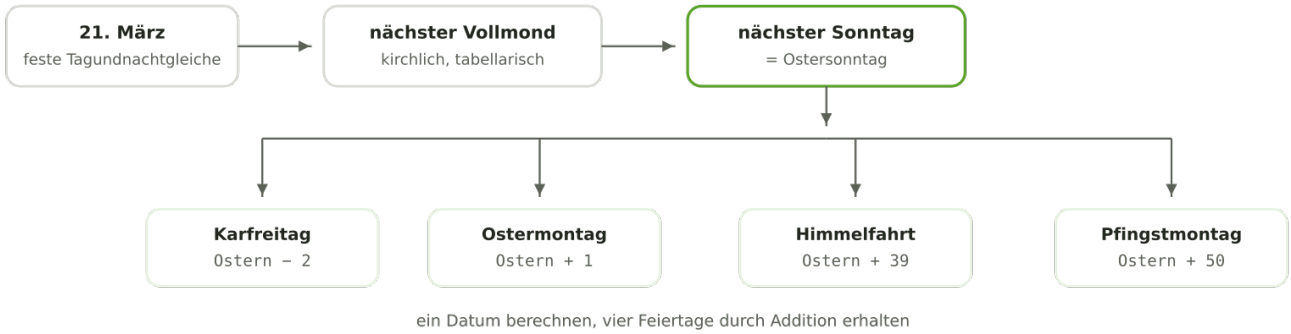
Die gute Nachricht: Das Ganze passt in etwa fünfzehn Zeilen. Dieser Beitrag setzt Ostern um, ergänzt die deutsche Regeltabelle, behandelt den Umstand, dass beide Länder Wochenendfeiertage völlig unterschiedlich handhaben, und führt beide Kalender zu einem zusammen — alles, ohne die Auswertungsschleife aus Teil 1

anzufassen. Genau das ist der Punkt: Eine gute Datenstruktur nimmt eine wirklich neue Regelart auf, ohne dass man sie neu schreiben muss.

Warum Ostern wandert

Ostern ist der erste Sonntag nach dem ersten kirchlichen Vollmond am oder nach dem 21. März. Jeder Halbsatz darin arbeitet: Der *kirchliche* Vollmond ist eine tabellarische Näherung statt einer astronomischen Beobachtung,

der 21. März ist ein fester Platzhalter für die Tagundnachtgleiche, und „der erste Sonntag danach" bindet das Ergebnis an einen Wochentag. Die Folge ist, dass Ostern zwischen dem 22. März und dem 25. April liegen kann.



Die Definition von Ostern verkettet drei Bedingungen — und vier deutsche Feiertage hängen am Ergebnis

Das Diagramm ist der Grund, warum sich die Mühe lohnt: Ostern ist die einzige wirklich schwierige Berechnung im deutschen Kalender, und sobald man sie hat, ist jeder der vier Feiertage ein einziger `IncDay`-Aufruf. Die Abstände sind fest — sie ändern sich nie von Jahr zu Jahr — der schwierige Teil ist also genau einmal gelöst.

Der Algorithmus, in fünfzehn Zeilen

Der übliche Weg, das gregorianische Osterdatum zu berechnen, ist der *anonyme gregorianische Algorithmus*,

meist Gauß zugeschrieben und von Butcher und Meeus in die Form unten gebracht. Es ist eine Folge von Ganzzahldivisionen und Restbildungen, die den Mondzyklus rekonstruiert, und er gilt für jedes Jahr des gregorianischen Kalenders. Ich gebe ihn so wieder, wie er auf Wikipedia dokumentiert ist, und behalte die traditionellen

Einbuchstaben-Variablennamen bei, weil sie ihn gegen jede andere veröffentlichte Implementierung prüfbar machen:

```
/// Ostersonntag (westlich/gregorianisch) für AYear.
/// Anonymer gregorianischer Algorithmus (Gauß / Meeus-Jones-Butcher).
function EasterSunday(const AYear: Word): TDate;
var
  a, b, c, d, e, f, g, h, i, k, l, m, Month, Day: Integer;
begin
  a := AYear mod 19;
  b := AYear div 100;
  c := AYear mod 100;
  d := b div 4;
  e := b mod 4;
  f := (b + 8) div 25;
  g := (b - f + 1) div 3;
  h := (19 * a + b - d - g + 15) mod 30;
  i := c div 4;
  k := c mod 4;
  l := (32 + 2 * e + 2 * i - h - k) mod 7;
  m := (a + 11 * h + 22 * l) div 451;
  Month := (h + l - 7 * m + 114) div 31;         // 3 = März, 4 = April
  Day   := ((h + l - 7 * m + 114) mod 31) + 1;
  Result := EncodeDate(AYear, Month, Day);
end;
```

Ich tue nicht so, als seien die einzelnen Schritte intuitiv — sie sind eine komprimierte Rekonstruktion des Meton-Zyklus, und die Zwischenwerte entsprechen nichts, was man als Datum wiedererkennen würde. Praktisch zählt, dass der Algorithmus deterministisch, abhängigkeitsfrei und gegen bekannte Werte testbar ist. Ostern 2026 ist der 5. April, 2027 der 28. März und 2024 war es der 31. März; wenn Ihre Implementierung diese drei reproduziert, haben Sie sie mit ziemlicher Sicherheit korrekt abgetippt.

> **Dies berechnet ausschließlich das *westliche* Osterdatum**
> Die meisten orthodoxen Kirchen berechnen Ostern nach dem julianischen Kalender, was in der Regel ein anderes Datum ergibt — 2026 ist das westliche Osterfest am 5. April, das orthodoxe am 12. April, und 2027 liegen sie fünf Wochen auseinander. Der Algorithmus oben ist der gregorianische, und auf ihm beruhen die deutschen Feiertage. Wenn Sie das orthodoxe Datum brauchen, ist das eine eigene Berechnung, kein
>
> Parameter dieser hier.

Die vier abgeleiteten Feiertage sind dann eine Frage der Tagesaddition — wofür `IncDay` aus `System.DateUtils` da ist:

```
function EasterOffset(const AYear: Word; const ADays: Integer): TDate;
begin
  Result := IncDay(EasterSunday(AYear), ADays);
end;
```

Die Abstände sollte man genau benennen, denn zwei davon werden traditionell als andere Zahlen zitiert. Christi Himmelfahrt ist *Ostern + 39*, nicht + 40, obwohl es der vierzigste Tag nach Ostern heißt — die alte Zählweise

rechnet den Ostersonntag selbst als Tag eins. Pfingstmontag ist aus demselben Grund Ostern + 50: Pfingsten ist inklusiv gezählt der fünfzigste Tag, liegt also auf Ostern + 49, und der Montag danach auf + 50.

Die deutsche Tabelle: neun Feiertage, denen alle Länder zustimmen

Das deutsche Feiertagsrecht funktioniert anders als das amerikanische Bundesmodell, und den Unterschied sollte man sauber treffen statt ihn zu verwischen. Feiertage werden von den *Ländern* festgelegt — jedes

Bundesland hat sein eigenes Feiertagsgesetz — es gibt also keine einheitliche Bundesliste. Was es stattdessen gibt, ist eine Schnittmenge: neun Tage, die alle sechzehn Länder unabhängig voneinander bestimmen. Das

sind die *bundeseinheitlichen Feiertage*, und sie sind der ehrliche Zuschnitt für einen allgemein verwendbaren Kalender.

Die eine echte Ausnahme ist der Tag der Deutschen Einheit, der durch Bundesrecht in Artikel 2 Absatz 2 des Einigungsvertrags von 1990 festgelegt ist. Alles andere auf der Liste ist Landesrecht, das zufällig überall übereinstimmt:

```
function GermanRules: TArray<THolidayRule>;
begin
  Result := [
    THolidayRule.Create('neujahr', 'Neujahrstag',
      function(const Y: Word): TDate begin Result := Fixed(Y, 1, 1); end),
    THolidayRule.Create('karfreitag', 'Karfreitag',
      function(const Y: Word): TDate begin Result := EasterOffset(Y, -2); end),
    THolidayRule.Create('ostermontag', 'Ostermontag',
      function(const Y: Word): TDate begin Result := EasterOffset(Y, 1); end),
    THolidayRule.Create('tag-der-arbeit', 'Tag der Arbeit',
      function(const Y: Word): TDate begin Result := Fixed(Y, 5, 1); end),
    THolidayRule.Create('christi-himmelfahrt', 'Christi Himmelfahrt',
      function(const Y: Word): TDate begin Result := EasterOffset(Y, 39); end),
    THolidayRule.Create('pfingstmontag', 'Pfingstmontag',
      function(const Y: Word): TDate begin Result := EasterOffset(Y, 50); end),
    THolidayRule.Create('tag-der-deutschen-einheit', 'Tag der Deutschen Einheit',
      function(const Y: Word): TDate begin Result := Fixed(Y, 10, 3); end),
    THolidayRule.Create('erster-weihnachtstag', '1. Weihnachtstag',
      function(const Y: Word): TDate begin Result := Fixed(Y, 12, 25); end),
    THolidayRule.Create('zweiter-weihnachtstag', '2. Weihnachtstag',
      function(const Y: Word): TDate begin Result := Fixed(Y, 12, 26); end)
  ];
end;
```

Neun Regeln, vier davon Oster-Abstände, und der Record aus Teil 1 hat die neue Regelart ohne eine einzige Änderung aufgenommen. Genau so soll der Entwurf wirken: `EasterOffset` ist einfach eine weitere `function(Year): TDate`, für die Tabelle strukturell nicht von `Fixed` zu unterscheiden.

> **Die Landesfeiertage, die diese Liste auslässt**
>
> Mehrere weithin begangene deutsche Feiertage fehlen hier bewusst, weil sie nicht bundeseinheitlich sind: Heilige Drei Könige (6. Januar) ist in drei Ländern Feiertag, Fronleichnam in Bayern und einigen weiteren, Allerheiligen in den überwiegend katholischen Ländern und der Reformationstag in weiten Teilen des Nordens und Ostens. Der Frauentag ist in Berlin und Mecklenburg-Vorpommern Feiertag. Jeder davon ist für Millionen Menschen ein echter freier Tag — sie sind nur nicht allgemeingültig, und ein Kalender, der sich als *die* deutsche Liste ausgibt und sie enthält, wäre für den größten Teil des Landes falsch. Wenn Sie
>
> Genauigkeit auf Länderebene brauchen, ist die natürliche Erweiterung ein Feld `States: TArray<string>` an `THolidayRule` und ein Filter in `ComputeYear`; die Datenstruktur nimmt das klaglos hin.

Die Tabelle für 2026 ausgeführt ergibt die neun Daten unten, wobei die vier von Ostern abgeleiteten sich im Frühjahr gruppieren, genau wie die Abstände es vorhersagen:

```
Neujahrstag                Thu, 01 Jan 2026
Karfreitag                 Fri, 03 Apr 2026
Ostermontag                Mon, 06 Apr 2026
Tag der Arbeit             Fri, 01 May 2026
Christi Himmelfahrt        Thu, 14 May 2026
Pfingstmontag              Mon, 25 May 2026
Tag der Deutschen Einheit  Sat, 03 Oct 2026
1. Weihnachtstag           Fri, 25 Dec 2026
2. Weihnachtstag           Sat, 26 Dec 2026
```

Karfreitag am 3. April und Ostermontag am 6. April umschließen den Ostersonntag am 5., und Christi Himmelfahrt ist ein Donnerstag — wie immer, da es Ostersonntag plus 39 Tage ist. Das sind nützliche Invarianten für einen Test.

Wo sich die beiden Länder wirklich unterscheiden: Wochenenden

Sehen Sie sich die deutsche Liste noch einmal an: Der Tag der Deutschen Einheit fällt 2026 auf einen Samstag, ebenso der 2. Weihnachtstag. In Deutschland ist das schlicht ein verlorener Feiertag — das deutsche Recht kennt keinen Mechanismus, einen auf ein Wochenende fallenden Feiertag zu verlegen, und Beschäftigte bekommen keinen Ersatztag. In Jahren wie diesem ist das ein wiederkehrendes Ärgernis in der öffentlichen Debatte.

Die Vereinigten Staaten machen das Gegenteil. Nach 5 U.S.C. § 6103(b) wird ein auf einen Samstag fallender

Feiertag am vorangehenden Freitag begangen und ein auf einen Sonntag fallender am folgenden Montag. Der Independence Day 2026, ein Samstag, wird von Bundesbediensteten also am Freitag, dem 3. Juli, begangen.

Das ist die eine Stelle, an der ein Länder-Flag wirklich gerechtfertigt ist, und der saubere Weg besteht darin, *Auftreten* und *Begehung* als getrennte Felder zu führen, statt das eine mit dem anderen zu überschreiben:

```
type
  TWeekendRule = (wrNoShift, wrShiftToNearestWeekday);

  THolidayCalendar = record
    CountryCode: string;
    Rules: TArray<THolidayRule>;
    Weekend: TWeekendRule;
  end;

function ObservedDate(const ADate: TDate; const ARule: TWeekendRule): TDate;
begin
  Result := ADate;
  if ARule = wrNoShift then
    Exit;
  case DayOfTheWeek(ADate) of
    DaySaturday: Result := IncDay(ADate, -1);   // begangen am Freitag davor
    DaySunday:   Result := IncDay(ADate, 1);    // begangen am Montag danach
  end;
end;
```

Beide Daten zu behalten heißt, dass Sie beide Fragen aus einer Tabelle beantworten können: „Ist heute Weihnachten?" nutzt das Auftretensdatum, „Ist das Büro geschlossen?" das begangene. Fassen Sie beide zu einem Feld zusammen, verlieren Sie die Möglichkeit, die erste Frage überhaupt zu stellen — und das wiegt schwerer, als es klingt, denn Jahrestags- und Altersberechnungen wollen das echte Datum, nicht das administrative.

> **Die Verschiebung kann einen Feiertag ins Vor- oder Folgejahr tragen**
>
> Neujahr 2028 fällt auf einen Samstag, die US-Bundespraxis begeht ihn also am Freitag, dem 31. Dezember 2027. Ein nur für 2027 erzeugter Kalender enthält ihn nicht, und einer für 2028 platziert ihn außerhalb des Jahres, zu dem er gehört. Wenn Sie einen strengen Kalenderjahresbereich erzeugen, berechnen Sie die Nachbarjahre mit und filtern anschließend — drei Jahre zu berechnen und zwei zu verwerfen ist billig, und dieser Fehler erzeugt eine Abweichung von einem Tag genau in dem Moment, in dem das Geschäftsjahr wechselt.

Beide Kalender zusammenführen

Mit beiden Tabellen ist ein kombinierter Kalender eine Faltung über die Länder, und das Dictionary aus Teil 1 übernimmt die Entdopplung:

```
type
  TObservedHoliday = record
    UID: string;
    Name: string;
    CountryCode: string;
    Occurs: TDate;      // das tatsächliche Kalenderdatum
    Observed: TDate;    // der freie Tag nach etwaiger Wochenendverschiebung
  end;

function ComputeCalendars(const ACalendars: TArray<THolidayCalendar>;
  const AYear: Word): TArray<TObservedHoliday>;
var
  Cal: THolidayCalendar;
  Rule: THolidayRule;
  Item: TObservedHoliday;
  List: TList<TObservedHoliday>;
begin
  List := TList<TObservedHoliday>.Create;
  try
    for Cal in ACalendars do
      for Rule in Cal.Rules do
      begin
        Item.UID         := Rule.UID;
        Item.Name        := Rule.Name;
        Item.CountryCode := Cal.CountryCode;
        Item.Occurs      := Rule.Compute(AYear);
        Item.Observed    := ObservedDate(Item.Occurs, Cal.Weekend);
        List.Add(Item);
      end;
    List.Sort(TComparer<TObservedHoliday>.Construct(
      function(const L, R: TObservedHoliday): Integer
      begin
        Result := CompareDate(L.Occurs, R.Occurs);
        if Result = 0 then
          Result := CompareStr(L.CountryCode, R.CountryCode);
      end));
    Result := List.ToArray;
  finally
    List.Free;
  end;
end;
```

`CompareDate` statt eines rohen `<`-Vergleichs ist die Gewohnheit aus Teil 1, die sich auszahlt — es vergleicht nur den Datumsanteil, sodass ein versehentlicher Zeitanteil an irgendeinem `TDate` die Sortierreihenfolge nicht durcheinanderbringen kann. Der Ländercode als zweites Kriterium hält die Ausgabe stabil, wenn zwei Feiertage auf dasselbe Datum fallen — womit wir bei der interessanten Kollision wären.

Führt man beide Kalender für 2026 aus, tragen zwei Daten Einträge aus beiden Ländern. Der 1. Januar ist der offensichtliche — New Year's Day und Neujahrstag sind überall derselbe Tag. Der andere ist der 25. Dezember, Christmas Day und 1. Weihnachtstag. Und es gibt einen feineren Beinahe-Zufall: **Memorial Day 2026 und**

Pfingstmontag 2026 fallen beide auf Montag, den 25. Mai — der letzte Montag im Mai trifft auf ein von

Ostern abgeleitetes Datum, ein Zusammentreffen, das 2026 gilt und 2027 bricht, wenn Pfingstmontag auf den 17. Mai und Memorial Day auf den 31. fällt.

Genau wegen dieser Kollision hat der Index in Teil 1 `AddOrSetValue` verwendet. Bauen Sie ein `TDictionary<TDate, string>` über einen zusammengeführten Kalender mit einfachem `Add`, dann löst 2026 am 25. Mai eine Duplikatschlüssel-Exception aus — in Produktion, in genau dem einen Jahr, in dem es passiert. Ein mehrwertiger Index ist die ehrliche Struktur, sobald mehr als ein Land im Spiel ist:

```
function MergedIndex(const ACalendars: TArray<THolidayCalendar>;
  const AYear: Word): TDictionary<TDate, TArray<string>>;
var
  H: TObservedHoliday;
  Existing: TArray<string>;
begin
  Result := TDictionary<TDate, TArray<string>>.Create;
  for H in ComputeCalendars(ACalendars, AYear) do
  begin
    if not Result.TryGetValue(H.Observed, Existing) then
      Existing := [];
    Result.AddOrSetValue(H.Observed, Existing + [H.CountryCode + ': ' + H.Name]);
  end;
end;
```

Auf `Observed` statt auf `Occurs` zu indizieren ist hier die richtige Voreinstellung, denn die Frage, die dieser Index beantwortet, lautet „Hat heute irgendwo jemand frei?" Tauschen Sie das Feld, wenn Sie die kalendarische Frage stellen — und dass ein einziger geänderter Bezeichner die Bedeutung sauber umstellt, ist die letzte Dividende dafür, beide Daten behalten zu haben.

In der Praxis: Werktage

Der Lohn für all das ist meist eine Werktagsberechnung, und die ist jetzt kurz genug, um sie nebenbei zu schreiben. Beachten Sie, dass sie Feiertage aus jedem Jahr braucht, das der Bereich berührt — das ist der praktische Grund, warum `ComputeYear` ein Jahr entgegennimmt und nicht einen globalen Cache hält:

```
function AddBusinessDays(const AStart: TDate; const ACount: Integer;
  const AHolidays: TDictionary<TDate, TArray<string>>): TDate;
var
  Remaining: Integer;
begin
  Result := DateOf(AStart);
  Remaining := ACount;
  while Remaining > 0 do
  begin
    Result := IncDay(Result, 1);
    if (DayOfTheWeek(Result) in [DayMonday..DayFriday]) and
       (not AHolidays.ContainsKey(Result)) then
      Dec(Remaining);
  end;
end;
```

Dass `DayOfTheWeek` nach ISO nummeriert ist, macht `[DayMonday..DayFriday]` zu einem lesbaren Bereich statt zu einem Rätsel — mit dem sonntagsbasierten `DayOfWeek` aus SysUtils würde die Arbeitswoche unschön über die Mengengrenze laufen. Eine Kleinigkeit, aber es ist der Unterschied zwischen einer Zeile, die man liest, und einer, die man entschlüsselt.

> **Wann Sie stattdessen zu einer Bibliothek greifen sollten**
>
> Diese Serie baut rund hundertzwanzig Zeilen, die Sie vollständig verstehen — für ein oder zwei Länder der richtige Handel. Er hört schnell auf, der richtige zu sein. Wenn Sie Regeln auf deutscher Länderebene, US-Bundesstaatsfeiertage oder irgendein drittes Land brauchen, sind die gepflegten Open-Source-Optionen die bessere Antwort: python-holidays und Nager.Date decken beide Untergliederungen in über hundert Ländern ab, und date-holidays tut dasselbe in JavaScript. Sie verfolgen außerdem das, was handgeschriebener Code nie tut: Regel*änderungen*, etwa die Aufnahme von Juneteenth 2021 oder einen einmaligen nationalen Feiertag für ein Staatsbegräbnis. Speziell für Delphi ist TZDB das Gegenstück für die Zeitzonenhälfte des Problems und ohnehin empfehlenswert.

Fazit

Zwei Beiträge, zwei Länder — und die Auswertungsschleife hat sich kein einziges Mal geändert.

- **Ostern ist die einzige schwierige Berechnung in beiden Kalendern**, und sie besteht aus fünfzehn Zeilen Ganzzahlarithmetik ohne Abhängigkeiten. Vier deutsche Feiertage sind dann schlichte Abstände davon.
- **Die Abstände sind +39 und +50**, nicht +40 und +49 — die traditionelle Zählweise schließt den Starttag ein, und ein Fehler um eins bleibt hier unbemerkt.
- **Die neun bundeseinheitlichen Feiertage sind eine Schnittmenge, keine Bundesliste.** Nur der Tag der Deutschen Einheit ist bundesrechtlich festgelegt; die anderen acht sind sechzehn übereinstimmende Landesgesetze.
- **Auftreten und Begehung sind verschiedene Daten.** Die USA verschieben Wochenendfeiertage auf den nächsten Werktag, Deutschland verschiebt nie etwas. Behalten Sie beide Felder, und Sie können beide Fragen beantworten.
- **Zwei Feiertage können auf dasselbe Datum fallen** — Memorial Day und Pfingstmontag kollidieren am 25. Mai 2026 — jeder Index über einen zusammengeführten Kalender muss also damit rechnen.

> Das Schwierige an einem Feiertagskalender ist nicht der Algorithmus, sondern die Modellierung. Bekommen Sie „ein Feiertag ist eine Funktion von einem Jahr auf ein Datum" richtig hin, und Ostern, Wochenendverschiebungen und ein zweites Land fügen sich ohne Neuschreiben ein.

Wenn Sie das Fundament unter alldem wollen — was ein `TDateTime` wirklich ist und die `System.DateUtils`-Funktionen, auf die sich diese beiden Beiträge durchgehend stützen — deckt das Tutorial Datum und Uhrzeit in Delphi es in zwei Teilen ab, einschließlich der Zeitzonen-Disziplin, die in dem Moment zählt, in dem Ihr Feiertagskalender von einem Client in einem anderen Land konsumiert wird.

> **Die Serie: Gesetzliche Feiertage in Delphi berechnen**
> Zwei Teile: 1. Regeln als Daten und der US-Bundeskalender 2. Ostern, der deutsche Kalender und zwei Länder zusammenführen **— Sie sind hier** Beide bauen auf dem Tutorial Datum und Uhrzeit in Delphi auf.

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 —