NEWS · JULY 7, 2026

Collections in Delphi und Spring4D

By Dr. Holger Flick



Öffnen Sie fast irgendein ernsthaftes Delphi-Projekt, und Sie finden überall dieselben zwei Freunde: `TList<T>` und `TDictionary<T, V>`. Sie sind gut. Sie haben sich ihren Platz verdient. Für einen riesigen Anteil der täglichen Arbeit sind sie wirklich alles, was Sie brauchen.

Doch hin und wieder stoßen Sie auf eine Problemform, für die die Runtime Library keinen Container hat — Sie möchten einen Schlüssel, der auf *viele* Werte abbildet, oder ein Dictionary, in dem Sie von *bei-*

den Seiten nachschlagen können, oder eine Queue, die still ihren ältesten Eintrag verwirft, wenn sie voll

ist, oder Sie möchten einfach `people.Where(...).OrderBy(...)` schreiben, wie Sie es in C# oder LINQ tun würden, und weitermachen. Das ist der Moment, in dem viele von uns zu Spring4D greifen — einer

Open-Source-Basisbibliothek (Apache-2.0) von Stefan Glienke, die seit 2009 still und stetig gereift ist und deren `Spring.Collections.*`-Units, meiner ehrlichen Meinung nach, zu dem Code gehören, mit dem man im gesamten Delphi-Ökosystem am angenehmsten arbeitet.

Ich möchte Ihnen die Tour geben, die ich mir vor Jahren gewünscht hätte: die ganze Breite dessen, was in diesem Werkzeugkasten steckt, mit echtem Code — und dann, genauso ehrlich, wo die *moderne* Delphi-RTL

so gut aufgeholt hat, dass Sie die Abhängigkeit vielleicht gar nicht brauchen. Beides ist gleichzeitig wahr, und genau das macht den Reiz aus.

Alles im Folgenden ist im tatsächlichen Spring4D-Quellcode verankert (die Unit `Spring.Collections` und ihre Geschwister, Copyright 2009–2024). Wo ich eine Zahl oder eine Behauptung nenne, verlinke ich sie. Wo ich Ihnen meine Meinung gebe, sage ich es dazu.

Zuerst das, was den Knoten platzen lässt: Alles ist ein Interface

Vor jeder konkreten Collection steht eine Designentscheidung, die das gesamte Erlebnis prägt — deshalb zeige ich sie, bevor ich irgendetwas anderes zeige.

In Spring4D erzeugen Sie keine *Klasse*, die Sie mit `Free` freigeben müssen. Sie bitten eine Factory um ein **Interface**, und Delphis Referenzzählung räumt für Sie auf, sobald es den Gültigkeitsbereich verlässt.

```
uses
  Spring.Collections;

procedure Demo;
var
  names: IList<string>;      // ein Interface, keine Klasse
begin
  names := TCollections.CreateList<string>;
  names.Add('Ada');
  names.Add('Grace');
  // kein try/finally, kein names.Free – das Interface ist referenzgezählt
end;
```

Diese `TCollections`-Factory (deklariert in `Spring.Collections.pas`) ist die Eingangstür zur gesamten Bibliothek: `CreateList`, `CreateDictionary`, `CreateSet`, `CreateStack`, `CreateQueue`, `CreateDeque`, `CreateMultiMap` und viele mehr, jeweils mit Overloads für Comparer, Anfangswerte und Ownership.

Hier ist das mentale Modell — denn genau dieser Unterschied lässt den Rest mühelos wirken.



RTL-Collections sind Klassen, die Sie besitzen und freigeben; Spring4D gibt Ihnen referenzgezählte Interfaces

Keiner der beiden Ansätze ist „richtig" — es sind unterschiedliche Philosophien, und beide haben treue Anhänger. Die explizite Ownership der RTL ist transparent und hat null Referenzzählungs-Overhead. Spring4Ds Interface-Modell beseitigt eine ganze Kategorie von `try/finally`-Boilerplate und die Lecks, die entstehen, wenn man es vergisst. Ich persönlich liebe das zweite Modell; die bestehenden Konventionen Ihrer Codebasis zählen hier aber mehr als meine Vorliebe.

Nun eine offene Beobachtung aus Jahren der Arbeit mit Delphi-Teams: **Viele langjährige Delphi-Entwickler haben Interfaces schlicht nie zu einem Teil ihres täglichen Werkzeugkastens gemacht.** Das ist keine Kritik — Delphis Objektmodell ist auch ohne sie vollkommen produktiv, und eine Menge exzellenter, ausgelieferter Software wurde von Entwicklern geschrieben, die instinktiv zu `TObject`-Nachkommen und `try/finally` greifen und nie mehr brauchten. Aber es bedeutet, dass sich Spring4Ds Interface-first-Design anfangs ungewohnt anfühlen kann — gerade weil es auf einem Sprachfeature aufbaut, das viele von uns übersprungen haben. Wenn das auf Sie zutrifft: Der Gewinn, sich mit Interfaces vertraut zu machen, reicht weit über diese eine Bibliothek hinaus — es ist eine der lohnendsten Fähigkeiten im modernen Delphi.

Demnächst: Interfaces, richtig erklärt

Interfaces verdienen mehr als einen Absatz, deshalb plane ich einen eigenen Beitrag dazu — was sie sind, warum Referenzzählung die Speicherverwaltung einfacher statt schwerer macht und wie eine Handvoll Muster (wie das, das Spring4D hier verwendet) Code aufräumen kann, den Sie seit Jahren pflegen. Wenn Interfaces immer der Teil von Delphi waren, an dem Sie nur höflich vorbeigenickt haben, wird dieser Beitrag genau für Sie geschrieben. Bleiben Sie dran.

Ein schönes Detail: Ownership auch für die *Elemente*

Bei Objekt-Collections kann Spring4D auch die Elemente freigeben, nicht nur den Container. `TCollections.CreateObjectList<T>(ownsObjects: True)` liefert eine `IList<T>`, die jedes Objekt freigibt, wenn es entfernt wird oder wenn die Liste selbst stirbt — ganz wie die `TObjectList<T>` der RTL, aber verpackt im selben referenzgezählten Interface. Dictionaries nehmen ein `TDictionaryOwnerships`-Set (`doOwnsKeys`, `doOwnsValues`) entgegen, sodass Sie Schlüssel, Werte, beides oder keines besitzen können.

LINQ in Delphi, wirklich

Das Feature, das mich als Erstes überzeugt hat und mich bis heute begeistert: `IEnumerable<T>` bringt einen vollständigen Satz von Query-Operatoren mit — Sie drücken also aus, *was* Sie wollen, statt eine Schleife von

Hand zu bauen.

Beginnen wir mit dem Alltagsfall: eine Liste filtern, sortieren und ein Feld herausziehen. Von oben nach unten gelesen sagt der Code genau, was er tut.

```
uses
  Spring.Collections;

var
  people: IList<TPerson>;
  names: IEnumerable<string>;
begin
  // ... people befüllen ...

  // Erwachsene, nach Name sortiert, projiziert auf nur den Namen
  names :=
    TEnumerable.Select<TPerson, string>(
      TEnumerable.OrderBy<TPerson, string>(
        people.Where(function(const p: TPerson): Boolean
          begin Result := p.Age >= 18 end),
        function(const p: TPerson): string
          begin Result := p.Name end),
      function(const p: TPerson): string
        begin Result := p.Name end);
end;
```

Sobald sich das flüssig liest, folgen die mächtigeren Operatoren demselben Muster. Hier ist Aggregation — Personen nach Stadt gruppieren, um etwa zu zählen, wie viele in jeder wohnen:

```
var
  people: IList<TPerson>;
  byCity: IEnumerable<IGrouping<string, TPerson>>;
  group: IGrouping<string, TPerson>;
begin
  // ... people befüllen ...

  byCity :=
    TEnumerable.GroupBy<TPerson, string>(people,
      function(const p: TPerson): string
        begin Result := p.City end);

  for group in byCity do
    WriteLn(group.Key, ': ', group.Count);   // z. B. "Berlin: 12"
end;
```

Die Operatoren verteilen sich auf zwei Zuhause, und es lohnt sich zu wissen, warum — denn das ist ein Fakt der Delphi-Sprache, keine Spring4D-Eigenheit. Methoden, die keinen neuen Typ einführen, liegen direkt auf `IEnumerable<T>` — `Where`, `Take`, `Skip`, `Distinct`, `Concat`, `Union`, `First`, `Any`, `All`, `Min`, `Max`, `Sum`, `Aggregate`, `ToArray` und mehr. Methoden, die einen neuen Ergebnis- oder Schlüsseltyp *einführen* (`Select<T, TResult>`, `GroupBy`, `OrderBy`, `Zip`, `SelectMany`), können in Delphi keine generischen Interface-Methoden sein und leben deshalb als statische Methoden auf `TEnumerable`. Genau deshalb liest sich `Where` oben fluent als `people.Where(...)`, während `OrderBy`, `Select` und `GroupBy` als `TEnumerable.OrderBy(...)` aufgerufen werden. Kennt man diese Aufteilung einmal, überrascht die API nicht mehr.

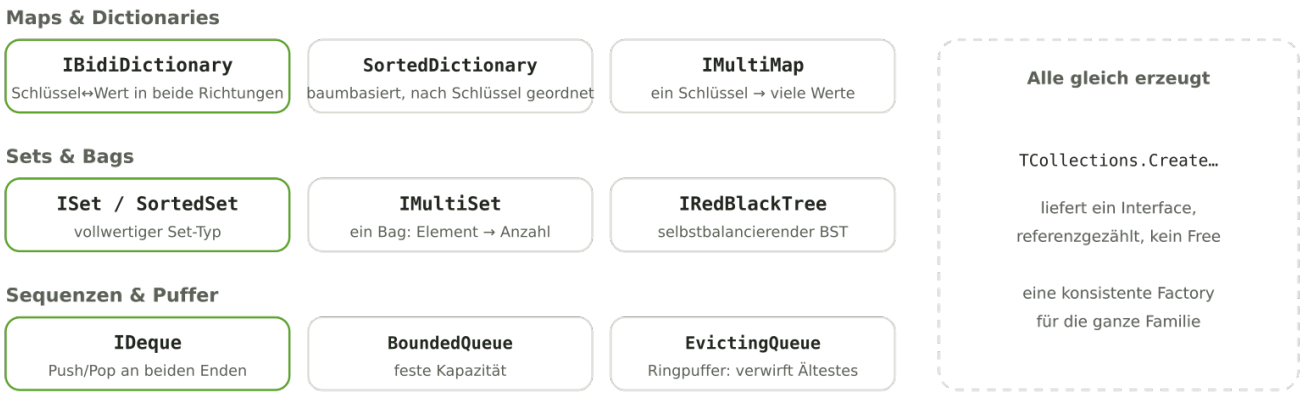
> **Eine kleine Ehrlichkeitsnotiz zur API-Oberfläche**
>
> Springs `IEnumerable<T>` hat `ToArray`, aber es gibt kein `ToList` — die Materialisierer sind `ToArray`, `TEnumerable.ToDictionary` und `TEnumerable.ToLookup`. Der „except"-Operator heißt `Exclude`, und Sortierung läuft über `OrderBy`/`OrderByDescending`/`Ordered` (ein `ThenBy` habe ich im Quellcode nicht gefunden). Nichts davon ist ein Mangel — es ist einfach die tatsächliche Form, und ich möchte lieber, dass Sie sie aus einer faktenbasierten Tour lernen, als dass die IDE Sie überrascht.

> **Alternativen, im Geiste einer fairen Tour**
>
> Spring4D ist nicht der einzige Weg zu Query-artigen Helfern in Delphi — System.Generics.Collections.TEnumerable bietet eine Handvoll eingebauter. Und wenn Sie sich jemals nach *schwergewichtigen* mengenbasierten Abfragen sehnen, ist das oft ein Signal, dass die Arbeit näher an die SQL-Engine Ihrer Datenbank gehört. Spring4Ds Stärke ist, dass es breit, schnell und alles an einem gut getesteten Ort ist.

Die Collections, die die RTL schlicht nicht mitliefert

Hier landet die Werkzeugkasten-Metapher wirklich. Über `IList` und `IDictionary` hinaus enthält Spring4D Container-Formen, die kein direktes Äquivalent in `System.Generics.Collections` haben. Das ist kein Seitenhieb auf die RTL — sie ist eine bewusst schlanke Standardbibliothek — es ist einfach echt mehr Vielfalt.

Lassen Sie mich die Familie ausbreiten, denn sie zusammen zu sehen ist die halbe Miete.



Spring.Collections liefert Container-Formen, die die RTL nicht von Haus aus hat

Einige davon sind jeweils einen Satz wert, denn erst der *Anwendungsfall* lässt sie einleuchten:

- `IMultiMap` — ein Schlüssel, der auf eine Sammlung von Werten abbildet (`CreateMultiMap`, `CreateHash-MultiMap`, `CreateTreeMultiMap` sowie sortierte Varianten). Perfekt für „alle Bestellungen dieses Kunden", ohne dass Sie Listen-von-Listen von Hand verwalten.
- `IBidiDictionary` — ein bidirektionales Dictionary, das Sie über Schlüssel *oder* Wert abfragen können. Praktisch für stabile ID-”-Name-Zuordnungen, bei denen Sie beide Richtungen brauchen.
- `IMultiSet` — ein Bag/Zähler, der mitführt, wie oft jedes Element vorkommt. Ein Worthäufigkeitszähler wird zum Zweizeiler.
- `IDeque` **und die begrenzten/verdrängenden Puffer** — doppelseitige Queues und Ringpuffer auf Basis eines gemeinsamen zirkulären Puffers. Eine `EvictingQueue`, die nur die letzten N Log-Zeilen behält, ist genau das, was Sie sonst neu erfinden und subtil falsch machen würden.
- `IRedBlackTree` **und sortierte Sets/Dictionaries** — echte selbstbalancierende Bäume, wenn Sie geordnete Iteration mit gutem Einfüge-/Nachschlageverhalten brauchen.

Hier der Zähler in der Praxis — genau die Sorte Code, die man mit echtem Vergnügen *nicht* von Hand schreiben muss:

```
var
  wordCount: IMultiSet<string>;
  word: string;
begin
  wordCount := TCollections.CreateMultiSet<string>;
  for word in Tokenize(document) do
    wordCount.Add(word);
  // wordCount['the'] ist jetzt die Anzahl der Vorkommen von 'the'
end;
```

Lebensdauer, Reihenfolge und Events, die Sie gratis bekommen

Ein paar weitere Annehmlichkeiten runden ab, warum sich der Werkzeugkasten stimmig anfühlt statt wie ein Haufen Einzelteile — lassen Sie mich sie gruppieren.

Erstens: **Die Einfügereihenfolge bleibt standardmäßig erhalten.** `TCollections.CreateDictionary<TKey, TValue>` liefert tatsächlich ein `IOrderedDictionary` zurück, sodass die Iteration Ihnen die Einfügereihenfolge zurückgibt — etwas, das das klassische `TDictionary` der RTL nie versprochen hat.

Zweitens: **Änderungsbenachrichtigungen sind eingebaut.** Collections stellen ein `OnChanged`-Event bereit (Dictionaries zusätzlich `OnKeyChanged` und `OnValueChanged`), das Aktionen wie `caAdded`, `caRemoved` und `caReplaced` meldet. Es gibt sogar Factories für Observable Lists (`CreateObservableList`) für UI-artige Binding-Szenarien.

```
var
  items: IList<string>;
begin
  items := TCollections.CreateList<string>;
  items.OnChanged.Add(
    procedure(Sender: TObject; const Item: string;
      Action: TCollectionChangedAction)
    begin
      if Action = caAdded then
        Log('added: ' + Item);
    end);
  items.Add('hello');   // löst den Handler aus
end;
```

Das ist das ganze Verkaufsargument für den Werkzeugkasten: breit, konsistent, referenzgezählt, beobachtbar und schnell (mehr zur Geschwindigkeit gleich). Nun lassen Sie mich genauso begeistert die andere Seite der Geschichte erzählen.

Und wo die moderne RTL alles ist, was Sie brauchen

Hier kommt der Teil, bei dem ich besonders fair sein möchte, denn er ist eine wirklich gute Nachricht für jeden Delphi-Entwickler: **Die Standardbibliothek hat einen weiten Weg zurückgelegt, und aktuell zu bleiben zahlt sich auf eine Weise aus, die nicht nur in sichtbaren Komponenten besteht.**

Wir neigen dazu, ein neues Delphi-Release danach zu beurteilen, was wir auf ein Formular ziehen können. Aber ein Großteil der wertvollsten Arbeit der letzten Releases fand *unter der Haube* statt — im Compiler und in der RTL. Die generischen Collections wurden wiederholt verfeinert: Delphi 10.3 Rio fügte `TryAdd` hinzu, brachte messbare Beschleunigungen bei Kernoperationen wie `Add` und `IndexOf` und machte `for-in`-Schleifen spürbar schneller. Jüngst hat Delphi 12 `TOrderedDictionary<K,V>` eingeführt und damit einfügegeordnete Iteration direkt in `System.Generics.Collections` gebracht — eine Fähigkeit, die früher ein Grund war, zu einer Bibliothek zu greifen. Und Delphi 13 Florence (erschienen September 2025) ist die aktuelle, moderne Basis, auf der diese ganze Diskussion steht.

Die ehrliche Zusammenfassung: Die Lücke hat sich stetig geschlossen, Release für Release.

> **Das Upgrade-Argument, klar ausgesprochen**
> Wenn Sie auf einem älteren Delphi sind und ein Upgrade abwägen, sind die RTL- und Compiler-Verbesserungen ein echter Teil des Werts — nicht nur die neuen visuellen Bausteine. Ein modernes `TDictionary` und ein eingebautes `TOrderedDictionary` sind die Art stiller Gewinne, die Alltagscode schneller und einfacher machen. Aktuell zu bleiben ist eines der besten „Features", die Sie Ihrer Codebasis geben können.

Für einen großen Anteil des Alltagscodes — eine `TList<T>` von Records, ein `TDictionary<Integer, TThing>` für Lookups, ein gewöhnliches `for-in` — gilt also: **Die RTL ist kein Kompromiss. Sie ist das richtige Werkzeug, ohne einzige zusätzliche Abhängigkeit.** Wenn Ihr Problem diese Form hat, können Sie die Drittbibliothek getrost weglassen und verlieren nichts, was zählt.

Performance: Was tatsächlich bekannt ist — fair, von beiden Seiten

Performance verdient Sorgfalt, denn hier lässt sich am leichtesten übertreiben — also halte ich mich an das, was dokumentiert ist, und kennzeichne den Rest.

Der stärkste veröffentlichte Datenpunkt auf der Spring4D-Seite stammt von seinem Autor Stefan Glienke, der die Neuentwicklung der Collections für Spring4D 2.0 beschreibt: Die 1.x-Collections waren, in seinen eigenen offenen Worten, langsam — "dead slow, especially on Win32" (todlangsam, besonders auf Win32) — weil fast jede Methode virtuell war. Version 2.0 entfernte den virtuellen Dispatch hinter den Interfaces und überarbeitete die Interna, und er berichtet in seinem Benchmark von **rund 4–7× Verbesserung gegenüber der RTL bei Dictionary-Lookups**. Das ist ein bemerkenswertes Ergebnis — und genau die Art Engineering, die einer Bibliothek ihren Ruf einbringt.

Zwei Portionen Ehrlichkeit gehören allerdings direkt neben diese Zahl.

> **Wo ein Benchmark aufhört zu verallgemeinern**
> Ein 4–7×-Ergebnis bei Dictionary-Lookups ist ein *spezifischer* Mikro-Benchmark (ein bestimmter Schlüsseltyp, eine bestimmte Trefferquote, und Spring4D 2.0 gegen die RTL der damaligen Zeit). Er ist real und beeindruckend — aber er ist kein pauschales „Spring4D ist bei allem 5× schneller" und sollte auch nicht so gelesen werden. Andere Operationen, Schlüsseltypen und Delphi-Versionen fallen anders aus. **Die einzige Zahl, die Ihre Entscheidung bestimmt, ist die, die Sie auf Ihrer eigenen Workload messen.**
>
> Beide Bibliotheken sind schnell genug, dass Sie den Unterschied im meisten Code nie spüren; wenn Sie in einem echten Hot Path stecken, profilen Sie ihn.

Die faire Lesart von beiden Seiten lautet: Spring4Ds Collections sind ernsthaft gut optimiert *und* die RTL hat sich weiter verbessert. Diese beiden Fakten widersprechen sich nicht. Wenn roher Durchsatz auf einer bestimmten Struktur Ihr Engpass ist, lohnt sich Spring4D unbedingt für einen Benchmark — und ebenso lohnt es sich, schlicht auf dem neuesten Delphi zu sein.

Also — wozu greifen Sie?

Lassen Sie mich die Tour zu einer praktischen Entscheidung zurückführen, denn „es kommt darauf an" ist nur nützlich, wenn ich Ihnen sage, *worauf* es ankommt. Dieser nächste Teil ist meine Meinung, geformt aus Jahren mit beiden; nehmen Sie ihn als Ausgangspunkt, nicht als Regel.

> **Eine Faustregel nach Passform (meine Sicht)**
> **Greifen Sie zu Spring.Collections, wenn die* Form *Ihres Problems dem entspricht, was nur diese Bibliothek bietet: Sie wollen LINQ-artige Abfragen inline; Sie brauchen eine MultiMap, ein bidirektionales Dictionary, ein MultiSet/Bag, eine Deque oder einen verdrängenden Ringpuffer; Sie wollen referenzgezählte Collections ohne* `try/finally`*; oder Sie wollen Änderungsbenachrichtigungen. In diesen Fällen spart sie Ihnen*

> *echten Code und echte Bugs. Greifen Sie zur RTL, wenn Ihre Anforderungen klar in ihrem Revier liegen: gewöhnliche Listen und Dictionaries, geordnete Iteration (Sie haben jetzt* `TOrderedDictionary`*), und Sie möchten dem Projekt lieber keine Abhängigkeit hinzufügen. Auf einem modernen Delphi deckt das eine* Menge *Code ab, und es ist eine völlig gute Antwort. Die entscheidende Frage ist nicht „was ist besser" — beide sind exzellent. Sie lautet: „Braucht mein Problem etwas, das nur eine der beiden hat?"* Wenn ja, wählt sich das Werkzeug von selbst. Wenn nein, gewinnt das, was schon im Projekt ist, durch Einfachheit.

Es gibt hier keine falsche Wahl, und — wichtig — keinen Grund für Dogmatismus. Viele gesunde Codebasen nutzen standardmäßig die RTL und holen Spring4D genau dort hinein, wo eine MultiMap oder ein verdrängender Puffer den Code offensichtlich besser macht. Beides zu mischen ist völlig in Ordnung.

Fazit

Spring.Collections ist ein wirklich reichhaltiger, gut konstruierter Werkzeugkasten, und ich greife nach wie vor gern hinein — aber die moderne Delphi-RTL hat die Lücke so weit geschlossen, dass die richtige Antwort endlich, sauber, eine Frage der *Passform* ist.

- **Die RTL ist exzellent für den Normalfall** und wird immer besser — `TryAdd`, schnellere Generics und Delphi 12s `TOrderedDictionary` sind echte Gewinne. Aktuell zu bleiben ist eines der besten Upgrades, das Sie machen können.
- **Spring4D glänzt, wo die RTL nicht hinreicht:** LINQ-artige Abfragen inline, MultiMaps, bidirektionale und sortierte Dictionaries, MultiSets, Deques, verdrängende Ringpuffer, referenzgezählte Lebensdauer und Change-Events — alles hinter einer konsistenten `TCollections`-Factory.
- **Bei der Performance: Respekt für beide.** Spring4D 2.0s Neuentwicklung ist beeindruckend schnell (der Autor berichtet 4–7× bei Dictionary-Lookups), und die RTL ist alles andere als langsam — aber nur *Ihr* Benchmark auf *Ihrer* Workload sollte über einen Hot Path entscheiden.
- **Und wenn der Interface-first-Stil neu für Sie ist: Das ist eine Chance, keine Mauer** — ein eigener Beitrag über Interfaces folgt bald, denn sie zahlen sich weit über diese eine Bibliothek hinaus aus.

> Zwei großartige Werkzeuge, eine ehrliche Frage: Braucht Ihr Problem etwas, das nur eines von beiden bietet? Wenn ja, wählt das das Werkzeug. Wenn nein, halten Sie es einfach — und so oder so gewinnt Delphi.

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 —