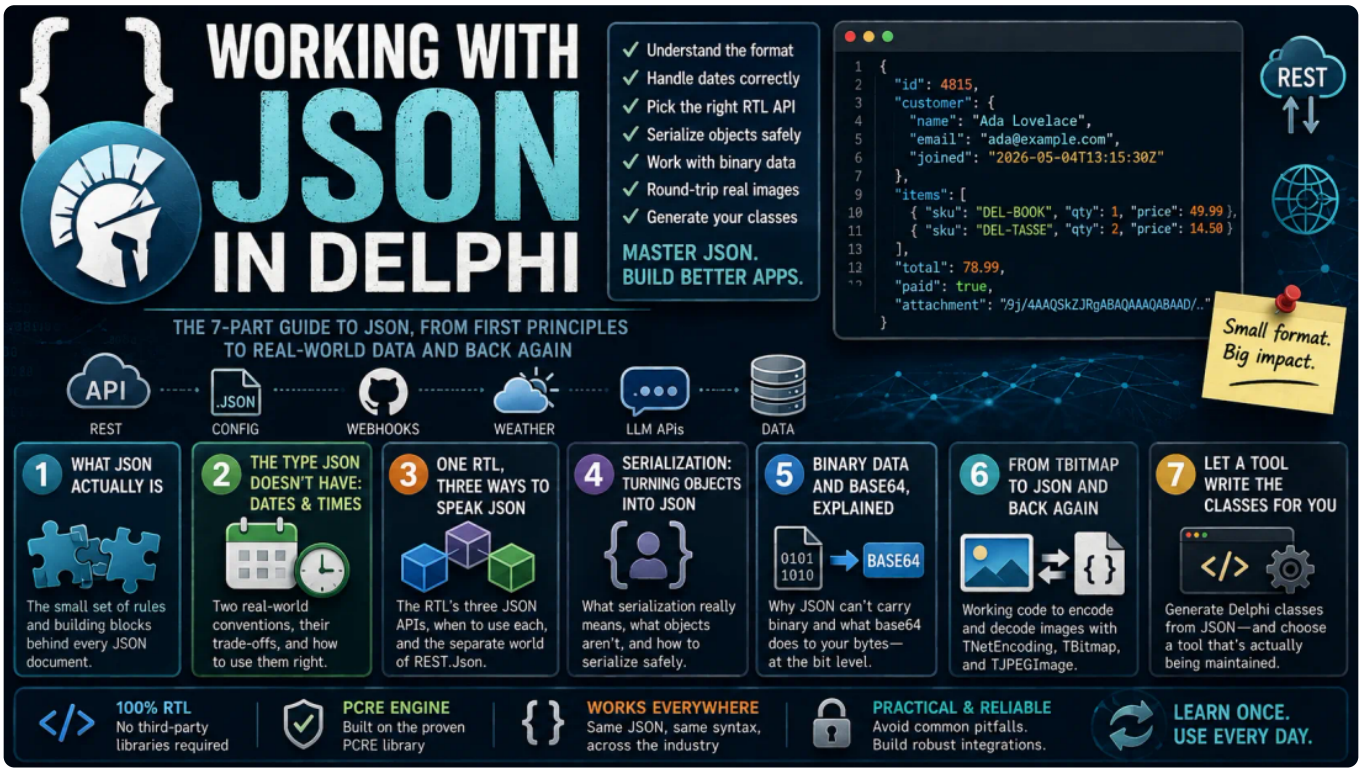
TUTORIAL · JULY 25, 2026

JSON mit Delphi

Ein siebenteiliger Kurs über JSON für Delphi-Entwickler — von dem, was JSON eigentlich ist, über Datumswerte, die drei JSON-APIs der RTL und die Serialisierung, bis zum Einbetten von Binärdaten und echten Bildern per base64 und schließlich zum Erzeugen von Delphi-Klassen direkt aus einem JSON-Dokument. Mit lauffähigem Delphi-Code und den Denkmodellen, die alles selbstverständlich machen.

By Dr. Holger Flick



Fast alles, was Ihre Delphi-Anwendung nach außen sagt, sagt sie in JSON. Der REST-Aufruf beim Zahlungsanbieter, die Konfigurationsdatei beim Start, der Webhook von GitHub, die Antwort eines Wetterdienstes oder eines LLM — all das kommt in demselben kleinen Textformat mit geschweiften und eckigen Klammern an, das Sie schon tausendmal getippt haben. Die meisten von uns haben es nebenbei gelernt: gerade genug, um das Feld herauszuholen, nie ganz genug, um sicher zu sein, was das Format eigentlich verspricht.

Dieses Tutorial ist der Kurs, der das behebt, ausgeliefert als siebenteilige Artikelserie. Er beginnt vor dem Code — mit dem, was JSON *ist*, und den zwei Typen, die es bewusst nicht hat — und greift erst dann zu Pascal,

sodass der Code, wenn er kommt, selbstverständlich statt magisch wirkt. Unterwegs klärt er die Dinge, die still und leise echte Nachmittage kosten: warum die RTL gleich *drei* JSON-APIs mitbringt, warum Serialisierung auf

eine Weise scheitert, die sich auf eine einzige falsche Annahme zurückführen lässt, und was base64 wirklich mit Ihren Bytes macht.

Lesen Sie die Teile der Reihe nach — jeder setzt genau dort an, wo der vorherige aufgehört hat, und der spätere Code stützt sich auf die früheren Denkmodelle.

Teil 1 — Was JSON eigentlich ist

Wir sind längst von JSON umgeben — hier ist, was es wirklich ist

Vor der ersten Zeile Pascal: was JSON wirklich ist, warum sich die ganze Branche darauf geeinigt hat, warum es immer wieder „das, was XML abgelöst hat" genannt wird, und aus welcher überraschend kleinen Menge von Regeln sein gesamtes Datenmodell besteht. Am Ende können Sie jedes JSON-Dokument ansehen, egal wie tief verschachtelt, und die wenigen Bausteine erkennen, aus denen es zusammengesetzt ist — das Denkmodell, auf dem jede Bibliothek dieser Serie aufbaut.

Teil 2 — Der Typ, den JSON nicht hat: Datum und Zeit

Der Datumstyp, den JSON nicht hat

In Delphi ist ein Datum ein echter Typ, den Compiler und Debugger verstehen. Serialisieren Sie eines nach JSON, verschwindet das alles — denn zu JSONs sechs Werttypen gehört kein Datum, also ist ein Datum in JSON gar kein Typ, sondern eine *Konvention*, auf die sich beide Seiten außerhalb des Formats einigen müssen.

Dieser Teil behandelt die zwei Konventionen, die die Welt tatsächlich verwendet, was jede kostet und wie Sie sie mit `System.DateUtils` korrekt erzeugen und einlesen — bevor die falsche Wahl eine Rechnung einen Tag zu früh datiert oder Ihre Zeitstempel ins Jahr 2033 springen lässt.

Teil 3 — Eine RTL, drei Wege, JSON zu sprechen

Eine RTL, drei Wege, JSON zu sprechen

Jetzt der Code — und eine Tatsache, die erfahrene Delphi-Entwickler überrascht: Die RTL gibt Ihnen nicht *eine* JSON-API, sondern **drei**, in verschiedenen Units, mit wirklich unterschiedlichen Stärken, und die meisten

begegnen nur der ersten. Das ist völlig in Ordnung — bis ein 300-MB-Export den Prozess mit sich reißt. Dieser Teil geht alle drei durch, sagt Ihnen, wann Sie zu welcher greifen, und entwirrt die zweite, völlig eigenständige JSON-Welt namens `REST.Json` in Ihrer `uses`-Klausel: woher sie kommt und wie Sie beide auf einen Blick auseinanderhalten.

Teil 4 — Serialisierung: Objekte in JSON verwandeln

Objekte sind keine Daten — Serialisierung wandelt Objekte in Daten

JSON Feld für Feld zu bauen, ist bei kleinen Nutzdaten ehrlich und bei einer Klasse mit zwölf Feldern mühsam. Der Wunsch — *verwandle einfach mein Objekt in JSON* — hat einen Namen, Serialisierung, und Delphi kann

das. Doch dieser Teil nimmt sich zuerst Zeit für die Bedeutung des Wortes, denn der Wunsch verbirgt eine Annahme, die nicht stimmt: dass ein Objekt *Daten* sei. Ist es nicht — es ist Daten plus Identität plus Verhalten

plus Referenzen plus, oft genug, ein Handle auf etwas, das nur existiert, während Ihr Prozess läuft. Fast jeder Serialisierungsfehler geht auf diese Verwechslung zurück, und hier lernen Sie, ihn kommen zu sehen.

Teil 5 — Binärdaten und base64, erklärt

Base64: Das, was alle benutzen und niemand erklärt

Der zweite Typ, den JSON nicht hat, und der schwierigere: Binärdaten. Ein JPEG, ein PDF, ein Zertifikat — keines davon ist ein String, eine Zahl, ein Boolean oder null, und keine Konvention macht aus einem JPEG eine Zahl. Die Antwort der Branche ist base64, und die meisten, die es korrekt verwenden, könnten nicht sagen, was es tut — „es komprimiert", „es verschlüsselt", beides falsch, das zweite gefährlich. Kein Delphi-Code hier: Dieser

Teil erklärt, was Binärdaten sind, warum JSON sie wirklich nicht tragen kann und was base64 auf Bit-Ebene mit Ihren Bytes macht — damit der Code in Teil 6 selbstverständlich wird.

Teil 6 — Von TBitmap nach JSON und zurück

Von TBitmap nach JSON und zurück

Die Belohnung, in lauffähigem Code. Kodieren und Dekodieren mit `TNetEncoding`, eine echte RTL-Falle, die stillschweigend JSON erzeugt, das Ihre Konsumenten ablehnen könnten, und vollständige Rundläufe mit echten Bildern — `TBitmap` und `TJPEGImage` — mit nichts außerhalb von RTL und VCL. Am Ende können Sie ein Bild in ein JSON-Dokument packen und dasselbe Bild wieder herausholen — und wissen genau, welche Schritte Daten verlieren können und welche nicht.

Teil 7 — Lassen Sie ein Werkzeug die Klassen schreiben

JSON zu Delphi-Klassen: Die populäre Fassung ist nicht die lebendige

Wenn Sie erst einmal wissen, wie die RTL JSON spricht, stellt sich die naheliegende Frage, warum Sie die DTO-Klassen überhaupt selbst tippen sollten — und es gibt ein kostenloses Werkzeug, das sie erzeugt: JSON-Dokument einfügen, kompilierbare Unit erhalten, mit Klassen, `REST.Json`-Attributen und fertiger Serialisierung. Fast jede Zeile seiner Ausgabe werden Sie aus Teil 3 und 4 wiedererkennen, einschließlich der zwei stillen Umgehungen, die es für Sie einbaut. Dieser Teil beginnt jedoch mit der kleinen grauen Zeile unter dem Repository-Namen, die die Geschichte verändert hat: Die Fassung, die alle finden, ist nicht die Fassung, die jemand pflegt — und die dreißig Sekunden Lesearbeit, die beide unterscheiden, sind mehr wert als das Werkzeug selbst.

Was Sie danach können

Arbeiten Sie sich durch alle sieben Teile, und JSON hört auf, ein Format zu sein, für das Sie Zauberformeln kopieren, und wird eines, das Sie wirklich verstehen:

- **Jedes JSON-Dokument lesen** und jeden Baustein benennen, aus dem es besteht.
- **Datumswerte über die Leitung bringen**, ohne jemandem den Tag oder das Jahrzehnt zu verschieben.
- **Die richtige JSON-API der RTL wählen** — und wissen, warum `REST.Json` eine eigene Welt ist.
- **Objekte sicher serialisieren** und dabei erkennen, welche Teile eines Objekts je wirklich Daten waren.
- **base64 erklären** — was es tut, was es kostet und was es *nicht* schützt.
- **Echte Bilder durch JSON schleusen** und dabei genau wissen, wo Daten verloren gehen können.
- **Ihre DTO-Klassen aus einem Beispieldokument erzeugen** — und beurteilen, bevor Sie sich darauf verlassen, ob das Projekt dahinter noch lebt.

> JSON ist ein kleines Format mit ein paar scharfen Kanten. Lernen Sie die Kanten einmal, und jede API, mit der Sie für den Rest Ihrer Laufbahn sprechen, wird einfacher.

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 —