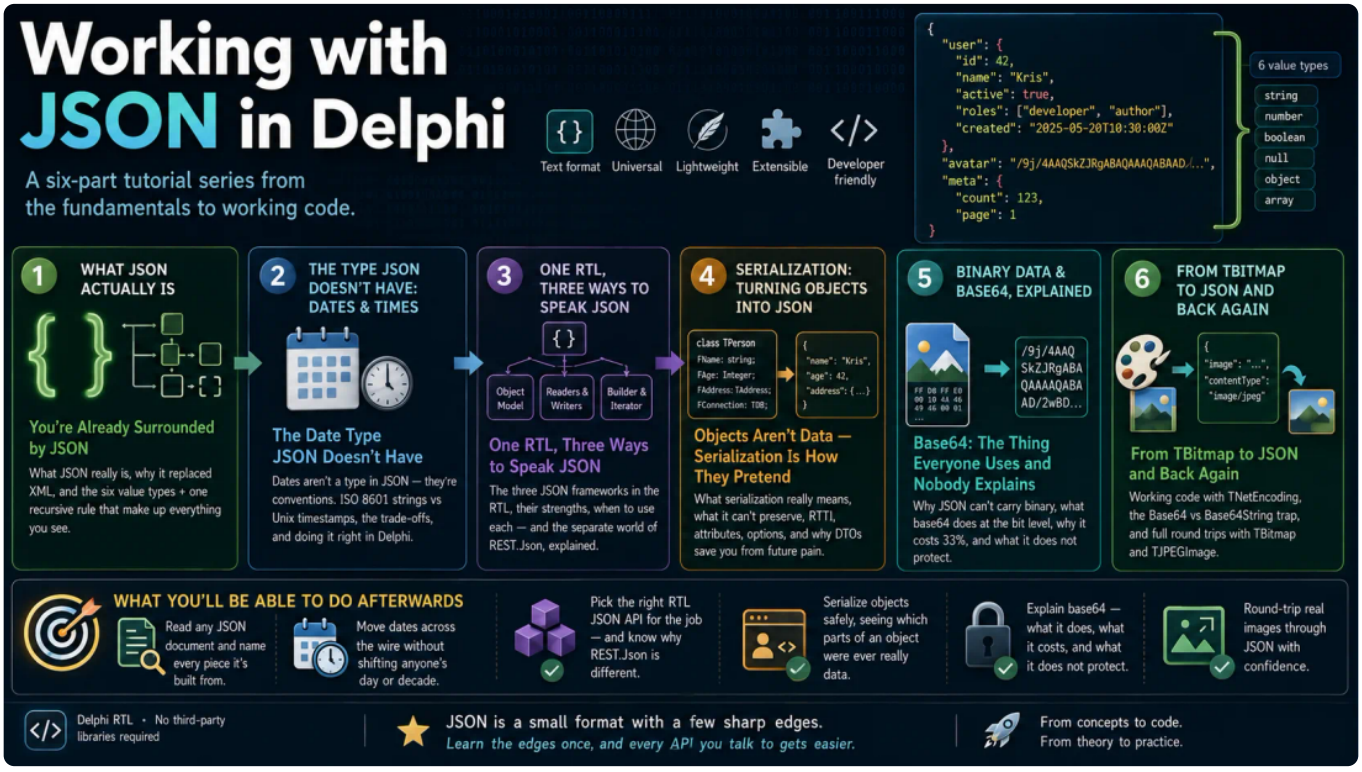
TUTORIAL · JULY 25, 2026

Working with JSON in Delphi

A six-part course on JSON for Delphi developers — from what JSON actually is, through dates, the RTL's three JSON APIs, and serialization, to putting binary data and real images into JSON with base64. Runnable Delphi code, and the mental models that make it all obvious.

By Dr. Holger Flick



Almost everything your Delphi app says to the outside world, it says in JSON. The REST call to a payment provider, the config file it reads on startup, the webhook from GitHub, the reply from a weather service or an LLM — all of it arrives as the same little text format with curly braces and square brackets that you've typed a thousand times. Most of us learned it by osmosis: enough to get the field out, never quite enough to be sure what the format actually promises.

This tutorial is the course that fixes that, delivered as a six-part article series. It starts before the code — with what JSON *is* and the two types it deliberately doesn't have — and only then reaches for Pascal, so that when the code arrives it looks inevitable rather than magical. Along the way it clears up the things that quietly cost real afternoons: why the RTL ships *three* different JSON APIs, why serialization breaks in ways that trace back to a single false assumption, and what base64 actually does to your bytes.

Read the parts in order — each one starts exactly where the previous one ended, and the later code leans on the earlier mental models.

Part 1 — What JSON actually is

You're Already Surrounded by JSON — Here's What It Actually Is

Before a single line of Pascal: what JSON really is, why the whole industry landed on it, why it keeps being called "the thing that replaced XML," and the surprisingly small set of rules its entire data model is built from. By the end you can look at any JSON document, however deeply nested, and see the handful of pieces it's assembled from — the mental model every library in this series is built to manipulate.

Part 2 — The type JSON doesn't have: dates and times

The Date Type JSON Doesn't Have

In Delphi a date is a real type the compiler and debugger understand. Serialize one to JSON and all of that evaporates — because JSON's six value types don't include a date, so a date in JSON isn't a type at all, it's a *convention* both ends have to agree on out of band. This part covers the two conventions the world actually uses, what each one costs, and how to produce and consume them correctly with `System.DateUtils` — before the wrong choice dates someone's invoice a day early or jumps your timestamps to 2033.

Part 3 — One RTL, three ways to speak JSON

One RTL, Three Ways to Speak JSON

Now the code — and a fact that surprises experienced Delphi developers: the RTL doesn't give you *one* JSON API, it gives you **three**, in different units, with genuinely different strengths, and most people only ever meet the first. That's fine right up until a 300 MB export takes the process down with it. This part walks through all three, tells you which to reach for, and untangles the second, completely separate JSON world called `REST.Json` sitting in your `uses` clause — where it came from and how to tell the two apart at a glance.

Part 4 — Serialization: turning objects into JSON

Objects Aren't Data — Serialization Is How They Pretend

Building JSON field-by-field is honest for small payloads and miserable for a class with twelve fields. The wish — *just turn my object into JSON* — has a name, serialization, and Delphi can do it. But this part spends real time on what the word means first, because the wish hides an assumption that isn't true: that an object *is* data.

It isn't — it's data plus identity plus behavior plus references plus, often, a handle to something that only exists while your process runs. Almost every serialization bug traces back to that confusion, and this is where you learn to see it coming.

Part 5 — Binary data and base64, explained

Base64: The Thing Everyone Uses and Nobody Explains

The second type JSON doesn't have, and the harder one: binary data. A JPEG, a PDF, a certificate — none of them is a string, a number, a boolean, or null, and no convention turns a JPEG into a number. The industry's answer is base64, and most developers who use it correctly can't tell you what it does — "it compresses it," "it encrypts it," both wrong, the second dangerous. No Delphi code here: this part is the explanation of what binary data is, why JSON genuinely can't carry it, and what base64 does to your bytes at the bit level — so the code in Part 6 is obvious.

Part 6 — From TBitmap to JSON and back again

From TBitmap to JSON and Back Again

The payoff, in working code. Encoding and decoding with `TNetEncoding`, one genuine RTL trap that silently produces JSON your consumers may reject, and complete round trips with real images — `TBitmap` and `TJPEGImage` — using nothing outside the RTL and VCL. By the end you can put a picture into a JSON document and get the same picture back out, and know exactly which steps can lose data and which can't.

What you'll be able to do afterwards

Work through all six parts and JSON stops being a format you copy incantations for and becomes one you actually understand:

- **Read any JSON document** and name every piece it's built from.
- **Move dates across the wire** without shifting anyone's day or decade.
- **Pick the right RTL JSON API** for the job — and know why `REST.Json` is a different world.
- **Serialize objects safely**, seeing which parts of an object were ever really data.
- **Explain base64** — what it does, what it costs, and what it does *not* protect.
- **Round-trip real images** through JSON with confidence about where data can be lost.

> JSON is a small format with a few sharp edges. Learn the edges once, and every API you talk to for the rest of your career gets easier.

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 —