TUTORIAL · AUGUST 6, 2026

Working with RTTI in Delphi

A three-part course on RTTI — Delphi's reflection — for Delphi developers: what run-time type information actually is and when it earns its place, reading properties, fields, and custom attributes at runtime, and invoking methods by name to build a CSV exporter that works on any class you'll ever write.

By Dr. Holger Flick

You have been using reflection for years. Every time the Object Inspector lists the published properties of a component you finished writing ten minutes ago, every time `REST.Json` turns one of your classes into JSON without being told anything about it, every time an ORM builds an `INSERT` statement from fields you invented last week — something is inspecting your code while the program runs. None of those tools were written with knowledge of *your* classes. They discover them.

The machinery behind all of it is **RTTI — Run-Time Type Information**, Delphi's implementation of the idea other ecosystems call *reflection*: code that examines and manipulates types it knew nothing about at compile time. The modern `System.Rtti` unit has shipped since Delphi 2010, and a striking number of developers have benefited from it daily without ever calling it directly. That's a shame, because once it clicks it opens a whole category of solutions that are impossible to write any other way — the ones where a single routine must work correctly on every class in your codebase, including the ones that don't exist yet.

This tutorial is that missing introduction, delivered as a three-part article series. It's a gentle tour rather than an API dump, and it is deliberately honest about the trade: reflection buys enormous flexibility with a little runtime cost and a lot of compile-time safety, so each part is equally clear about where *not* to reach for it.

Read the parts in order — each one starts exactly where the previous one ended, and the finale only works because the first two did their job.

Part 1 — What reflection is, and the two records it all starts from

What Is RTTI? Delphi's Reflection, Part 1

It opens with a question that sounds impossible: can a program inspect its own code while it's running? The answer turns on a single shift in perspective — the difference between a property name that is *compiled in* and a property name that is merely a **string**, arriving from a config file, a JSON key, or a database column. That indirection is the entire superpower, and this part builds the two things you need to exploit it: `TRttiContext`, your session with the reflection system (a record you nevertheless `Create` and `Free`, for a reason worth understanding before you write the first line), and `TValue`, the typed box that holds a value of *any* type while remembering exactly which type it is — and which turns out to be genuinely useful on its own, even if you never reflect on anything. It also does the thing most reflection tutorials skip: it names, concretely, the three situations where reaching for RTTI is the wrong call.

Part 2 — Reading a type: properties, fields, and attributes

RTTI Part 2: Reading Types, Properties, and Attributes

This is where the abstract idea becomes something you can watch work. It builds a routine that prints every property of any object — name, type, and current value — and that mentions no class anywhere in its body; it takes a plain `TObject` and describes `TCustomer`, `TButton`, or something you'll write next year. Then it turns the operation around: setting a property when the property's *name* arrives at runtime as data, guarded properly, in about a dozen lines. The second half reaches the feature that makes modern Delphi frameworks feel declarative — **custom attributes**, the `[Bracketed]` annotations you can attach to a property and read back at runtime.

By the end you've built an ORM's mapping layer in miniature, and you know why a class can declare its own database columns without a single line of per-class mapping code. There's a typed shortcut that removes the manual casting, and a clear-eyed section on what an attribute actually is when nothing reads it.

Part 3 — Calling methods by name, and a tool worth shipping

RTTI Part 3: Invoking Methods — and a Real Tool

If reading a property by name was a small magic trick, this is the whole show: calling `DoThing` on an object when the string `'DoThing'` only shows up at runtime — from a script, a config file, a plugin, a message off the wire. That single capability is how command dispatchers, scripting bridges, and test runners work, and one of `Invoke`'s overloads goes further still, in the exact way every dependency-injection container relies on. The part is honest about the safety net you give up in exchange, and names the error you'll meet when you get it wrong. Then it spends its second half building the capstone: a **generic object-to-CSV exporter** that produces a header row and one line per object for *any* list of *any* class, in roughly forty lines, using nothing but the three parts of the series — including an attribute that lets a class keep a sensitive property out of the export entirely.

What you'll be able to do afterwards

Work through all three parts and the machinery behind Delphi's serializers, ORMs, and DI containers stops being a black box:

- **Explain what RTTI is** — and recognize the shape of problem it solves, so you know a reflection job when you see one.
- **Open and close a reflection session correctly**, with clear rules about which objects you own and which you must never free.
- **Use** `TValue` to hold and pass a value of any type — including as a modern, type-safe alternative to `Variant` in code that has nothing to do with reflection.
- **Walk any object's properties and fields**, read and write them by name, and know when properties are the right choice and when raw storage is.
- **Define and read custom attributes**, so your classes can declare how a framework should treat them instead of you maintaining a mapping table.
- **Invoke methods — and constructors — by name**, and guard those calls when the name comes from outside your program.
- **Decide honestly when not to reflect**, and reach for a proven library instead of hand-rolling one.

> Reflection lets your program treat *types themselves* as data: read (`GetValue`), write (`SetValue`), and call (`Invoke`) any member by name. Write the walker once, and it works on every class you'll ever add.

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 —