TUTORIAL · AUGUST 6, 2026

RTTI mit Delphi

Ein dreiteiliger Kurs über RTTI — Delphis Reflection — für Delphi-Entwickler: was Laufzeit-Typinformationen wirklich sind und wann sie sich lohnen, wie Sie Properties, Felder und eigene Attribute zur Laufzeit auslesen, und wie Sie Methoden über ihren Namen aufrufen, um daraus einen CSV-Export zu bauen, der mit jeder Klasse funktioniert.

By Dr. Holger Flick



Sie setzen Reflection seit Jahren ein. Jedes Mal, wenn der Objektinspektor die published Properties einer Komponente auflistet, die Sie vor zehn Minuten fertiggestellt haben; jedes Mal, wenn `REST.Json` eine Ihrer Klassen in JSON verwandelt, ohne je etwas über sie erfahren zu haben; jedes Mal, wenn ein ORM ein `INSERT` aus Feldern zusammenbaut, die Sie letzte Woche erfunden haben — dann untersucht etwas Ihren Code, während das Programm läuft. Keines dieser Werkzeuge wurde in Kenntnis *Ihrer* Klassen geschrieben. Sie entdecken sie.

Die Maschinerie dahinter heißt **RTTI — Run-Time Type Information**, Delphis Umsetzung dessen, was andere Ökosysteme *Reflection* nennen: Code, der Typen untersucht und manipuliert, von denen er zur Compilezeit nichts wusste. Die moderne Unit `System.Rtti` gibt es seit Delphi 2010, und erstaunlich viele Entwickler profitieren täglich davon, ohne sie je direkt aufzurufen. Das ist schade, denn sobald der Groschen fällt, öffnet sich eine ganze Klasse von Lösungen, die sich anders gar nicht schreiben lassen — nämlich überall dort, wo eine einzige Routine mit jeder Klasse Ihres Projekts korrekt umgehen muss, auch mit denen, die es noch gar nicht gibt.

Dieses Tutorial ist die fehlende Einführung, als dreiteilige Artikelserie. Es ist eine ruhige Führung und keine API-Auflistung — und es benennt den Handel offen: Reflection erkauft enorme Flexibilität mit etwas

Laufzeitkosten und viel Compilezeit-Sicherheit. Deshalb sagt jeder Teil ebenso klar, wo Sie *nicht* danach greifen sollten.

Lesen Sie die Teile der Reihe nach — jeder setzt genau dort an, wo der vorige endet, und das Finale funktioniert nur, weil die ersten beiden ihre Arbeit getan haben.

Teil 1 — Was Reflection ist, und die beiden Records, mit denen alles beginnt

Was ist RTTI? Delphis Reflection, Teil 1

Der Einstieg ist eine Frage, die zunächst unmöglich klingt: Kann ein Programm seinen eigenen Code untersuchen, während es läuft? Die Antwort hängt an einem einzigen Perspektivwechsel — dem Unterschied zwischen einem Property-Namen, der *einkompiliert* ist, und einem Property-Namen, der bloß ein **String** ist, der aus einer Konfigurationsdatei, einem JSON-Schlüssel oder einer Datenbankspalte kommt. Genau diese Indirektion ist die ganze Superkraft, und dieser Teil baut die zwei Bausteine, die Sie dafür brauchen: `TRttiContext`, Ihre Sitzung mit dem Reflection-System (ein Record, den Sie trotzdem mit `Create` und `Free` behandeln — aus einem Grund, den man vor der ersten Zeile Code verstanden haben sollte), und `TValue`, die typisierte Box, die einen Wert *jedes* Typs aufnimmt und sich dabei genau merkt, welcher Typ das war — und die sich auch ganz für sich allein als nützlich erweist, selbst wenn Sie nie etwas reflektieren. Und der Teil tut, was die meisten Reflection-Einführungen auslassen: Er benennt konkret die drei Situationen, in denen RTTI das falsche Werkzeug ist.

Teil 2 — Einen Typ auslesen: Properties, Felder und Attribute

RTTI Teil 2: Typen, Properties und Attribute auslesen

Hier wird aus der abstrakten Idee etwas, dem Sie bei der Arbeit zusehen können. Es entsteht eine Routine, die jedes Property eines beliebigen Objekts ausgibt — Name, Typ und aktueller Wert — und die in ihrem Rumpf keine einzige Klasse erwähnt; sie nimmt ein schlichtes `TObject` entgegen und beschreibt damit `TCustomer`, `TButton` oder etwas, das Sie erst nächstes Jahr schreiben. Danach wird die Operation umgedreht: ein Property setzen, wenn dessen *Name* erst zur Laufzeit als Datum ankommt — sauber abgesichert, in gut einem Dutzend Zeilen. Die zweite Hälfte erreicht das Feature, das moderne Delphi-Frameworks so deklarativ wirken lässt: **eigene Attribute**, jene `[Eckigen]` Annotationen, die Sie an ein Property hängen und zur Laufzeit wieder auslesen können. Am Ende haben Sie die Mapping-Schicht eines ORM im Kleinen gebaut und wissen, warum eine Klasse ihre Datenbankspalten selbst deklarieren kann, ohne eine einzige Zeile Mapping-Code pro Klasse. Es gibt eine typisierte Abkürzung, die das manuelle Casten überflüssig macht — und einen nüchternen Abschnitt darüber, was ein Attribut eigentlich ist, solange es niemand liest.

Teil 3 — Methoden über ihren Namen aufrufen, und ein Werkzeug zum Mitnehmen

RTTI Teil 3: Methoden aufrufen — und ein echtes Werkzeug

War das Lesen eines Property über seinen Namen ein kleiner Zaubertrick, ist das hier die ganze Vorstellung: `DoThing` auf einem Objekt aufrufen, wenn der String `'DoThing'` erst zur Laufzeit auftaucht — aus einem Skript, einer Konfigurationsdatei, einem Plugin, einer Nachricht von der Leitung. Genau darauf beruhen Command-Dispatcher, Scripting-Brücken und Test-Runner, und eine der Überladungen von `Invoke` geht noch einen Schritt weiter — exakt auf die Art, von der jeder Dependency-Injection-Container lebt. Der Teil ist ehrlich, welches Sicherheitsnetz Sie dafür aufgeben, und benennt den Fehler, den Sie sehen, wenn Sie es falsch

machen. Die zweite Hälfte baut dann das Schlussstück: einen **generischen CSV-Export**, der für *jede* Liste *jeder* Klasse eine Kopfzeile und eine Zeile pro Objekt erzeugt, in rund vierzig Zeilen, ausschließlich mit den drei Bausteinen der Serie — inklusive eines Attributs, mit dem eine Klasse ein sensibles Property ganz aus dem Export heraushält.

Was Sie danach können

Arbeiten Sie alle drei Teile durch, und die Maschinerie hinter Delphis Serialisierern, ORMs und DI-Containern ist keine Blackbox mehr:

- **Erklären, was RTTI ist** — und die Art von Problem erkennen, die es löst, sodass Sie eine Reflection-Aufgabe auf den ersten Blick sehen.
- **Eine Reflection-Sitzung korrekt öffnen und schließen**, mit klaren Regeln darüber, welche Objekte Ihnen gehören und welche Sie niemals freigeben dürfen.
- `TValue` **einsetzen**, um einen Wert beliebigen Typs zu halten und weiterzureichen — auch als moderne, typsichere Alternative zu `Variant` in Code, der mit Reflection gar nichts zu tun hat.
- **Properties und Felder jedes Objekts durchlaufen**, sie über ihren Namen lesen und schreiben, und wissen, wann Properties die richtige Wahl sind und wann der rohe Speicher.
- **Eigene Attribute definieren und auslesen**, damit Ihre Klassen selbst deklarieren, wie ein Framework sie behandeln soll, statt dass Sie eine Mapping-Tabelle pflegen.
- **Methoden — und Konstruktoren — über ihren Namen aufrufen** und diese Aufrufe absichern, wenn der Name von außerhalb Ihres Programms kommt.
- **Ehrlich entscheiden, wann Sie nicht reflektieren** und stattdessen zu einer erprobten Bibliothek greifen.

> Reflection lässt Ihr Programm *Typen selbst* als Daten behandeln: jedes Member über seinen Namen lesen (`GetValue`), schreiben (`SetValue`) und aufrufen (`Invoke`). Sie schreiben den Walker einmal — und er funktioniert mit jeder Klasse, die Sie je hinzufügen.

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 —