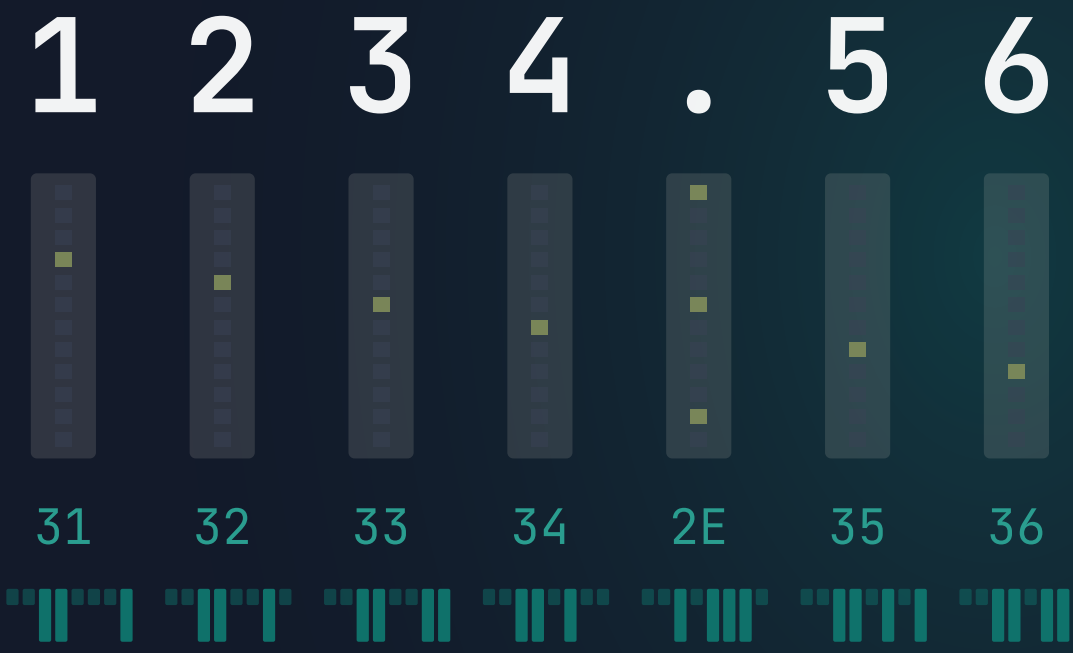


THE DATA SERIES, FROM TALLY TO TOKEN

What a Value Means

Types, Encodings and Formats

A stored value is usable only when its reader has the rules to interpret it: the type and its precision, the encoding, the unit, the time basis, the identity, what absence means and who defines its codes. The bits travel well; the rules often do not. This book follows one night across an insurer, a hospital and a hotel group, value by value, and gives a CTO the Type Contract: seven tested lines that make meaning travel with every field, to every reader, including the model.

Juan Cardena

Enterprise AI & Agentic Systems Architect · blog.jcardena.com

CONTENTS

BEFORE YOU START

Reader's Brief: One Night, Seven Lines

PROLOGUE

The Patient Who Arrived Before the Fire

PART I · THE VALUE

01 What a Value Is

02 Where the Contract Lives

PART II · KINDS OF VALUE

03 Integers and Floats

04 Money and Decimals

05 Text

06 Time

07 Identity: Keys and Identifiers

08 Codes, Units and Enumerations

09 Place

10 Absence

PART III · FORMATS

11 Records as Text: Fixed Width, CSV, XML and JSON

12 Schemas That Travel

13 Industry Messages

14 Values at Rest: Columnar Files and Archives

PART IV · THE MODEL READER AND THE WORK

15 The Model Reads and Writes Values

16 Type Contracts in 90 Days

CLOSING

Epilogue: The Contract Travels or the Meaning Does Not

APPENDICES

A The Type Contract: Template, Tests and Mappings

B Format Decision Table

C Failure Library

D Glossary

E Sources

BEFORE YOU START

Reader's Brief: One Night, Seven Lines



This book is about what it takes for a stored value to mean something to its next reader. It follows one night through fourteen numbered hops, each the move of one value, and gives you one tool, the Type Contract, to write down the rules a value needs. This brief tells you the story, the tool, what the book leaves to its sister books, and where to start for your role.

The night, in one paragraph

On Sunday 2 November 2025 clocks on the US East Coast fell back from 02:00 to 01:00, so every local time from 01:00 to 01:59 happened twice. In the first pass of that hour a kitchen fire set off fire panel 2 in a hotel. In the second pass an injured guest arrived at a hospital emergency department. Eight days later an insurer's claims-triage agent compared the two times as text and flagged the injury claim: treatment recorded 22 minutes before the fire alarm. Read as bare clock times, 01:20 is 22 minutes 10 seconds before 01:42:10. As instants, the guest arrived 37 minutes 50 seconds after the alarm, about 38 minutes. Every company, person, key and time in the story is invented and illustrative; every standard and practice is real and sourced.

Three companies, all illustrative

- **The insurer:** the mutual fire insurer founded in 1898, met in *From Tally to Token*. It writes the hotel's commercial package, policy CP 0041-7730, and pays the guest's liability claim.
- **The health system:** regional hospitals and clinics with HL7 v2 and FHIR inside. It releases records to the insurer only under the patient's written authorisation, as PDF documents.
- **The hotel and retail group:** about 60 hotels and 400 stores, with an EU booking site. The hotel of the night is in a US East Coast city, zone America/New_York.

The seven lines of the Type Contract

Line	It says	Its test at the boundary
1 Type, precision and rounding	Logical type, precision, scale, rounding rule, range	The value fits without silent rounding
2 Encoding and serialization	Character set, byte order, text form	The bytes decode with no replaced characters
3 Unit or currency	UCUM unit or ISO 4217 code	No number arrives without its unit
4 Time basis	Instant, wall time or date; zone; clock source	An instant without an offset is held
5 Identity scope	Which entity, which key system	The key is valid in the named system and resolves to a record there
6 Absence	What missing, null, empty and zero mean	Only the listed forms of absence pass
7 Authority	Code system and version, owner, allowed readers	Codes validate against the named edition

A value that fails a line is held at the boundary, not passed on. Chapter One explains each line; Chapter Sixteen builds the tests.

Borders

- Representations in general, Mars Climate Orbiter, packed decimal, the copybook and Y2K: *From Tally to Token*.
- Bits, binary arithmetic, words and file systems: *From Bit to Pixel*. Pixels, colour and fonts as rendering: *From Code to Glass*.
- Data contracts as an operating model and stages of trust: *From Bronze to Gold*.
- Platforms, table formats and exit; CSV and spreadsheets as desk culture; embeddings and retrieval: covered in depth later in this series.

Reading paths by role

Role	Read first	The decision you leave with
CTO	Prologue, Chapters 1, 6, 15, 16	Which boundary fields get a Type Contract this quarter
Chief data officer	Chapters 1, 2, 8, 10, 16, Appendix A	Who owns each code table and each company standard
Data or AI architect	Chapters 6, 11, 12, 14, 15, Appendix B	Which format per job, and what the AI reader receives with every value
CISO	Chapters 5, 11, 13, 15	Where parsing and encoding are attack surface

ERA CARD

Reader. You, with values that cross systems faster than their rules do.

Security. A value read under the wrong rules can pass every access control and still do harm.

Cost. Seven lines per boundary field, written once; the costed errors of the night took hours of staff time.

AI. A model that lacks the rules may infer plausible ones instead of stopping.

Monday rule: pick one time field and one money field your agents read, and ask whether the reader receives their seven lines.

PROLOGUE

The Patient Who Arrived Before the Fire



THE ERAS OF THE RECORD

On Monday 10 November 2025 an insurer's claims-triage agent, an AI assistant that summarises new claim files for the people who handle them, read the file for a hotel fire and flagged the injury claim of one guest for review. Its reason was short and specific: "medical treatment recorded 22 minutes before the fire alarm". A patient treated before the fire started is what a fraud reviewer looks for. The flag was wrong. The guest arrived at the emergency department 38 minutes after the alarm. On Tuesday 11 November a claims handler corrected the flag, using a station record the agent never saw and the "EST" the extraction had dropped. By then the flag was already a record of its own. This prologue walks it back to the values it came from.

The night, and why the flag sounded right

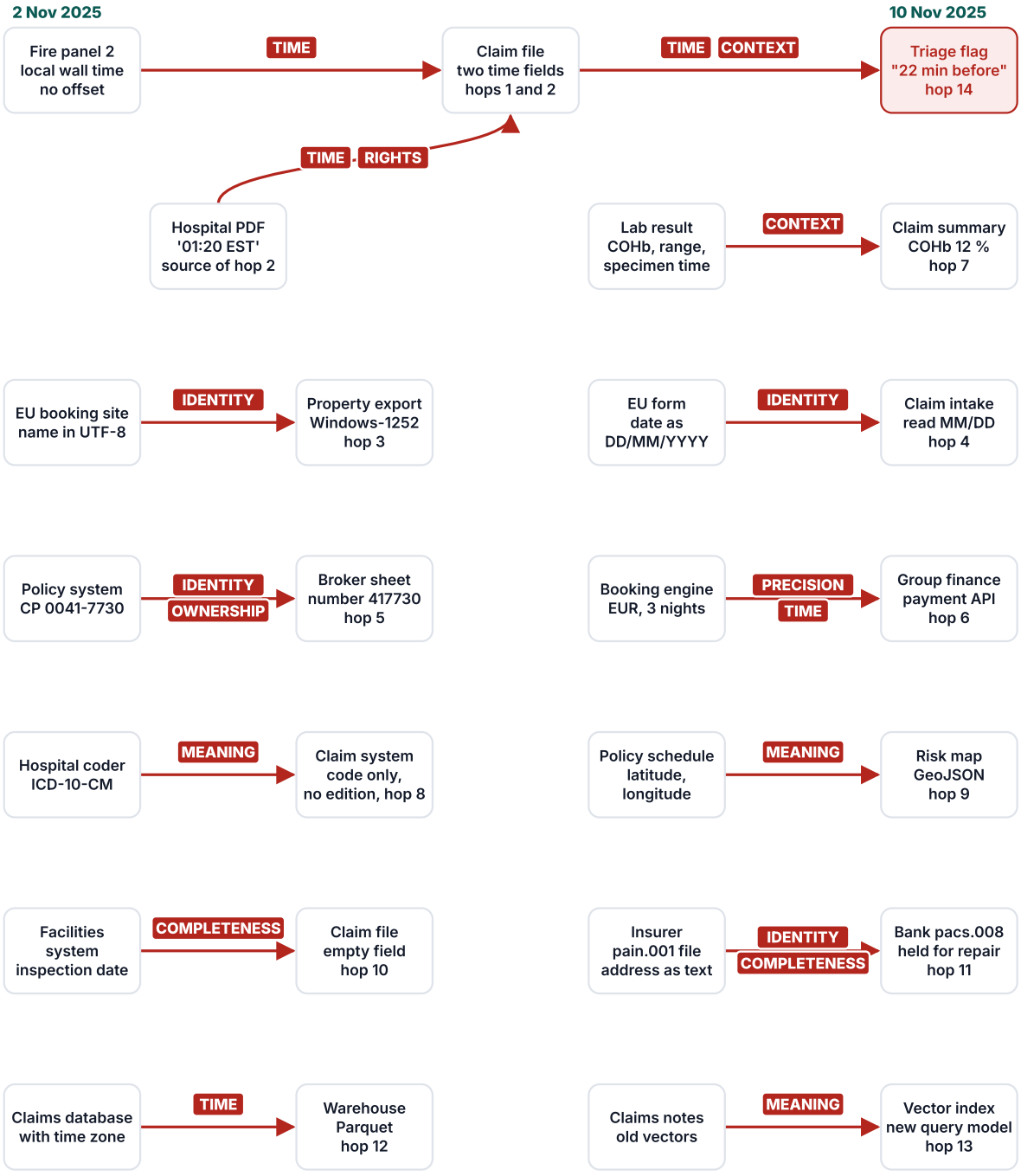
The night is an invented composite, and so is everyone in it. The companies are the three of this book: an insurer, a health system, and a hotel and retail group. The guest, named here Işıl Yılmaz, is an invented person. The standards, laws and practices the story touches are real, and each chapter sources them.

The fire happened on Sunday 2 November 2025, the night US daylight saving time ended. Federal law, 15 U.S.C. 260a, ends it at 2 a.m. on the first Sunday of November. At 02:00 Eastern Daylight Time clocks in the hotel's zone, America/New_York, went back to 01:00 Eastern Standard Time, so every local time from 01:00:00 to 01:59:59 happened twice. A kitchen fire set off fire panel 2 in the first pass of that hour. The guest, treated for smoke inhalation, reached the health system's emergency department in the second.

Eight days later the claim file held two time fields. The alarm read 2025-11-02 01:42:10. The arrival read 2025-11-02 01:20. Neither said which pass of the hour it belonged to. Read as plain clock times, 01:20 comes 22 minutes 10 seconds before 01:42:10, and the agent wrote that, rounded to 22 minutes, in fluent prose, with a review status that a person would act on. The hospital field was the arrival time; the word "treatment" was the agent's own paraphrase, a second slip.

FIGURE P.1

One night, fourteen hops: the main line (hops 1, 2 and 14) along the top, eleven more below, each with what it lost



Top row: the main line. Red arrows and chips: what each hop lost, by loss code.

The flag of 10 November 2025 sits at the end of the main line; the alarm time (hop 1) and the arrival time (hop 2, entering from below) are the two values it compared, and below them hop 7 sits beside the hospital source, with the other ten hops two to a row.

Author's diagram based on the book's running example, One Night, Every Value; companies, people, keys and times are illustrative.

Walking back: hops 14, 2 and 1

A **hop** is any move or change of a value between people, systems or formats. The night produced fourteen numbered hops, each the move of one value. Figure P.1 shows all fourteen. Three of them form the **main line** along the top: hops 1 and 2 carry the two time values, and hop 14 derives the flag from both. Read it backwards.

Hop 14, the flag. The agent compared the two time fields and wrote a sentence and a review status into the claim file: a new value, written by a model. It cited neither field's time basis, because neither field had one. Whatever the agent got wrong, it inherited from hops 1 and 2, and the flag now has a lineage of its own that the next reader can quote.

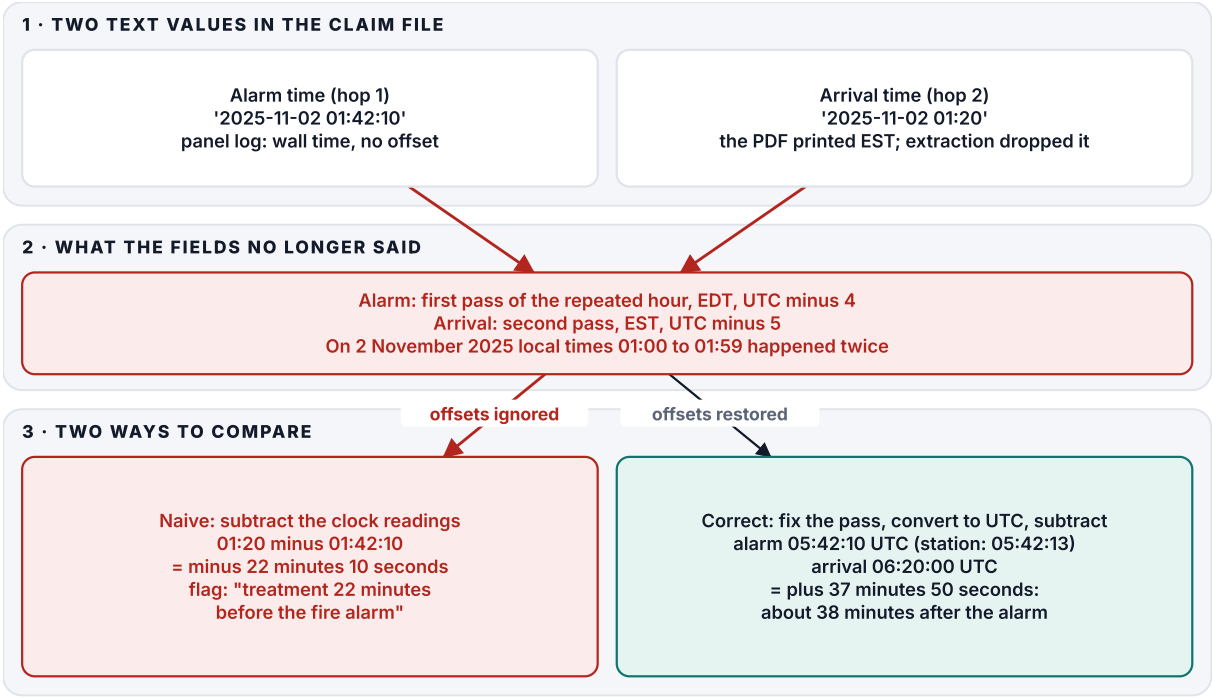
Hop 2, the arrival time. The hospital's registration system recorded the arrival with its offset, in HL7 v2 and then in FHIR inside the hospital. No feed connects the hospital to the insurer. The records travelled under the guest's written authorisation, which federal rules at 45 CFR 164.508 require to include "a description of the information to be used or disclosed" and "a description of each purpose of the requested use or disclosure". They arrived as PDF documents, and the arrival printed as 11/02/2025 01:20 EST. The insurer's document-extraction service read the page and filled the field with 2025-11-02 01:20. It kept the clock reading and dropped "EST". It also loaded every page into general claim fields, and the authorisation's description and purpose did not travel with those fields, so the hop lost a right as well as a time.

Hop 1, the alarm time. Fire panel 2 logs local wall time with no offset and no zone name. The facilities vendor's nightly export sent the event as text, 2025-11-02 01:42:10, and the claim file imported it on request. Inside the repeated hour that text names two instants an hour apart. The export said neither which pass nor which zone. Nothing in the claim file could tell 01:42 EDT from 01:42 EST.

The record that settles it. The handler on 11 November found a third clock. The hotel's central monitoring station received the alarm signal at 05:42:13 UTC, on its own synchronised clock. That receipt fits the panel's 01:42:10 only in the first pass, 05:42:10 UTC, three seconds earlier; in the second pass the panel's reading would fall almost an hour after the station had already received the alarm. The "EST" printed on the hospital's page fixes the arrival at 06:20 UTC, to the minute the record gives. Measured from the panel's reading, which is the event itself (the station decides only the pass), the arrival came 37 minutes and 50 seconds after the alarm (37 minutes 47 seconds from the station's receipt); this book rounds it to 38 minutes. The handler corrected the flag and recorded the correction and its evidence beside it.

FIGURE P.2

How "22 minutes before" was computed: two clock readings compared as if they were the same kind of time



The agent subtracted two clock readings from different passes of the repeated hour; once the station record fixes the alarm's pass, the two values converted to UTC put the arrival 37 minutes 50 seconds after the panel's alarm reading.

Author's diagram based on the book's running example; times computed with the IANA time zone database for America/New_York.

What the walk back shows

Figure P.2 puts the error in three steps. Two text values reached the claim file. Each had lost its offset, one at birth and one at extraction. The agent compared the clock readings. Once the station record restores the alarm's pass, the two values compared as instants in UTC give the right answer.

No clock digit was altered. No system crashed. Two kinds of time, a wall-clock reading and an instant, were treated as one, and a reader with no way to tell them apart made the simplest comparison the text permitted. The panel never recorded an offset, so no amount of reasoning recovers it from the panel's log. The alarm's pass had to come from an independent alarm-side record, the station's clock; the printed "EST" separately fixed the arrival.

> *The agent's subtraction was correct. It calculated with values that had lost the rules for reading them.*

A good agent could have noticed that 2 November 2025 was a fall-back night and asked which pass each time belonged to. Nothing in its instructions required that. A model may supply the plausible reading instead of stopping, unless the system around it says otherwise. The rules for reading a value must travel with it, and the agent must be told to stop when they are missing.

The main line is three of the fourteen hops. The same night lost a guest's name to a character set, a date of birth to day-month order, a policy number to a spreadsheet clean-up, a refund's dollars to a rate date, a carboxyhaemoglobin result to its context, a diagnosis code to its edition, a hotel to a swapped coordinate pair, an inspection date to an empty field, a payment to a bank's address and character-set rules, a warehouse's times to a UTC flag, and a search index to a model upgrade. Each chapter of Parts II and III takes one kind of value and its hop.

What this book gives you

The tool is the **Type Contract**: seven lines for every field that crosses a boundary, each with a test. Type, precision and rounding; encoding and serialization; unit or currency; time basis; identity scope; absence; authority. For the alarm time, line 4 would have said: an instant, with its offset recorded at the event, zone America/New_York, from a panel clock synchronised by NTP, a source nobody had recorded for this panel. The test at the claim file's door would have held 2025-11-02 01:42:10 as ambiguous, before any agent read it.

ERA CARD

Reader. A panel, a hospital system, an extraction service and an agent each read the times; only the last wrote a judgement about a person.

Security. The extraction loaded every page into general claim fields without the authorisation's description and purpose; scope is a value that must travel too.

Cost. Two hours of a handler's time and a guest's trust, for an offset nobody wrote down (illustrative).

AI. The agent filled a missing time basis with the plausible one; the contract must make it ask instead.

Monday rule: find every time field your agents compare, and check that each one carries its offset or is refused.

PART I · THE VALUE

CHAPTER ONE

What a Value Is



THE ERAS OF THE RECORD

At 01:42:10 on 2 November 2025, by its own clock, fire panel 2 wrote a line in its event log. Its nightly export later delivered the line to the insurer's claim file as nineteen characters: 2025-11-02 01:42:10. Every character arrived intact. A checksum would have passed, a parser would have accepted it, and a database would have stored it in a timestamp column without complaint. Yet on that night the text names two moments an hour apart, and nothing in it says which. The representation was perfect, and it did not identify one instant. This chapter separates those two ideas, adds a third, meaning, and names the rules a reader needs to get from one to the next. The decision it changes: **every field that crosses a boundary travels with a seven-line Type Contract, and each line has a test that a value must pass before it is used**.

Representation, value and meaning

Three words do the work in this book, and they are easy to blur.

A **representation** is the physical form a reader receives: bytes on a disk, characters in a file, holes in a card, glyphs on a page. The alarm's representation is nineteen characters of text. Representations can be copied, checked and compared byte for byte, and modern systems are very good at moving them without damage.

A **value** is what the representation stands for once a reader applies the right rules: a number, an instant, a person's name, an amount of money. Applied to 2025-11-02 01:42:10, the rules "this is local wall time in America/New_York, first pass of the repeated hour" give one instant, 05:42:10 UTC. The rules "second pass" give another, 06:42:10 UTC. The same representation, two values.

Meaning is what the value is about: which thing in the world, for which purpose, under whose authority. The instant 05:42:10 UTC is a bare point on the time line until someone says it is the alarm event of panel 2, at this hotel, as logged by the panel's clock, and that the central monitoring station's record wins if the two disagree.

Each step needs rules that the representation does not contain. Data people have known this for a long time. William Kent's *Data and Reality*, first published by North-Holland in 1978, says in its preface that file organisations, indexes, hierarchies, networks and relational models "are all just maps, being poor artificial approximations of some real underlying terrain". A few lines later he puts it in five words: "Data structures are artificial formalisms." The book is still the best short argument that a record and the thing it describes are different objects.

The standards world reached the same place by another road. ISO/IEC 11179, the international standard for metadata registries, maintained by the joint ISO and IEC subcommittee on data management and interchange, describes its own scope as "the semantics of data, the representation of data and the registration of the descriptions of that data". Part 1 of the standard, its framework, is candid about the limit: names, definitions and datatypes give a registered item meaning, but "the depth of this meaning is limited, because names and definitions convey limited information about the object".

So this book claims no discovery. Its claim is narrower and more practical. The representation travels well. The rules often do not. And the newest reader, a language model, does something earlier readers rarely did: when the rules are missing, it may supply plausible ones instead of stopping. What follows is the Type Contract introduced in *From Tally to Token*, here in full and with a test for each line. That book surveyed how one value looked in every era's representation; this one asks what each kind of value needs so that its next reader gets the value and not just the bytes.

One more definition, because the word is overloaded. In this book a **type** is the whole set of rules a reader needs, not only the storage type a database or programming language declares. A column declared TIMESTAMP has a storage type. Whether it holds an instant or a wall-clock reading, in which zone, from which clock, is also part of its type in this broad sense. The Type Contract writes that broad type down.

The seven lines, each with a test

A **Type Contract** is a short, structured statement attached to one field that crosses a boundary: from one system, company, format or reader to another. It has seven lines. Each line says something a reader would otherwise have to guess, and each has a test that runs where the value enters, so that a value which fails is held, not passed on.

Line	It says	A test at the boundary
1 Type, precision and rounding	The logical type; precision and scale; the rounding rule and where it applies; the valid range	Write 1234.565 into a field with two decimals: the boundary applies the named rule or refuses, never rounds silently
2 Encoding and serialization	Character set, byte order, the text form of numbers and dates	The bytes decode under the declared character set with no replacement characters, and a name such as Işıl survives a round trip
3 Unit or currency	A unit code (UCUM for measurements) or a currency code (ISO 4217); "not applicable" when there is none	A number that needs a unit and arrives without one is held
4 Time basis	Instant, wall time, date or duration; the zone by its IANA name; the clock source and how it is synchronised	2025-11-02 01:42:10 with no offset, declared an instant, is held as ambiguous
5 Identity scope	Which entity the value belongs to, and in which key system	The key matches the pattern and check digit of the named system and resolves to a record there; a key from another system is refused
6 Absence	What missing, null, empty and zero each mean for this field	An empty string where the contract says "never empty" is held, not read as "none"
7 Authority	The code system and its version; the owner; who may receive the value	A code validates against the named edition, and the receiver is on the allowed list

Line 1 deliberately puts rounding next to precision. Two systems can agree that an amount has two decimals and still disagree about the cents, because one rounded each line and the other rounded the total. Line 4 is the line the night broke, and Chapter Six is given to it. Line 5 is about the thing, not the format: a correctly formatted policy number that points at the wrong policy passes every syntax check, which is why its test also looks the key up. Line 7 joins two kinds of authority that are usually managed apart, the authority that defines what a code means and the authority that decides who may read it, because a value that reaches the wrong reader under the right definition is still a failure.

A line may not apply. The alarm time has no unit, and its contract says "not applicable" rather than leaving the line blank, so that a reader can tell a deliberate answer from a forgotten one. That is line 6 applied to the contract itself.

Figure 1.1 fills in the seven lines for the running example, the alarm time of hop 1, as the export sent it and as it would send it under a contract.

FIGURE 1.1

A Type Contract, before and after: the alarm time

Line of the Type Contract	Before: the export sent '2025-11-02 01:42:10'	After: the export sends '2025-11-02T01:42:10-04:00'
1 Type, precision and rounding	text; no type stated	instant, to the second
2 Encoding and serialization	UTF-8 assumed, not stated	RFC 3339 text with offset, UTF-8
3 Unit or currency	not applicable	not applicable
4 Time basis	no offset, no zone; in the repeated hour, two instants	offset recorded at the event; zone America/New_York; panel clock to be NTP-synchronised
5 Identity scope	unstated: which panel, which hotel	alarm event of panel 2 at the hotel
6 Absence	unstated: empty or missing?	missing = panel offline; never empty
7 Authority	unstated: no owner, no rule on conflict	owner: facilities manager; the central station record wins on conflict

The same alarm, exported first as bare text and then under a contract; the after column assumes the panel records its offset at the event, and only the timestamp travels in the text, while the other lines travel as field metadata beside it.

Author's diagram based on the book's running example (hop 1, illustrative); the seven lines are the Type Contract introduced in From Tally to Token.

Read the left column for what the claim file's next reader had to supply. The type was assumed; the encoding was assumed; the time basis was missing entirely. The right column needs no new technology. RFC 3339, the internet profile of date and time text, already requires an offset in every date-time: its grammar has no form without one. A panel clock can be given a time source with standard equipment. The facilities manager already owned the panel. What was missing was a place where these facts were written down, next to the value, in a form the next reader could check.

Typed storage is not typed meaning

A common answer to all this is that modern formats are typed. They are, and that is real progress. It is also less than it looks.

Take a money column in a warehouse file. The Apache Parquet specification defines DECIMAL as "arbitrary-precision signed decimal numbers of the form unscaledValue * 10^(-scale)", where scale is the number of digits to the right of the point and precision the maximum number of digits in the unscaled value. A column declared DECIMAL(18,2) can be stored as a 64-bit integer, since the specification allows int64 for a precision of up to 18. The value 1234.56 is stored as the integer 123456 with scale 2. It is exact. No binary rounding touches it. Any reader of the file can find the precision and scale in the file's own schema, beside the data.

Now ask the questions a finance reader asks. Euros or dollars? Rounded per line or once on the total, half-up or half-even? Converted at which rate, published on which date? Does 0.00 mean nothing is owed, or that the amount was never computed, and does null mean something different? Which booking does the amount belong to? Who owns the rounding rule? The DECIMAL section of the specification answers none of these questions, and was never meant to. Avro's decimal logical type, defined in the Avro specification by the same unscaled-times-ten-to-the-minus-scale formula with a required precision and an optional scale, is in the same position. Figure 1.2 sets out the gap line by line.

FIGURE 1.2

What DECIMAL(18,2) knows and does not

Line	The reader asks	Does DECIMAL(18,2) say?	Where the answer must come from
1 Type, precision and rounding	Is it exact, and how many digits?	Partly: exact, up to 18 digits, 2 after the point. It names no rounding rule	The contract names the rule and the point where rounding happens
2 Encoding and serialization	How are the bytes laid out?	Yes, inside the file: an unscaled integer, for example INT64, times 10 to the minus 2	The file format itself
3 Unit or currency	Euros, dollars or yen?	No	An ISO 4217 code, in a column or in the field's metadata
4 Time basis	Converted at which rate, on which date?	No	A rate date and source, stated per value or per field
5 Identity scope	Which booking, which line, which key system?	No	The key columns and the key system they belong to
6 Absence	Does 0.00 mean nothing owed, and does null mean unknown?	No	The contract's absence line
7 Authority	Who owns the rule, and who may receive the value?	No	An owner, the governing rule and its version, the allowed readers

A Parquet DECIMAL(18,2) column fully answers one of the reader's seven questions and part of another; the rest must travel beside it (line 4 is asked of a converted amount).

Source: Apache Parquet, Logical Types (DECIMAL); author's analysis.

The same holds for time. A Parquet TIMESTAMP carries a flag, isAdjustedToUTC. When it is true, the specification defines the value as time elapsed since the Unix epoch, an instant; when false, as date and time fields "in a local timezone, regardless of what specific time zone is considered local": a wall-clock reading with no zone stored. That flag is a real piece of line 4, written into the file. But it records only what the writer asserted. A loader that writes a local wall time with the flag set to true produces a file that is perfectly typed and wrong, and a reader that does not check it against the source trusts it. Chapter Fourteen follows exactly that hop.

The lesson is not that types are useless. Storage types answer the questions a machine needs answered to store and compute: how many bytes, which arithmetic. Contracts answer the questions a reader needs answered to use the value. A strong type system makes the first set reliable and leaves the second set where it was: in someone's head, a wiki page, or nowhere.

Three objections, answered

"This is the data dictionary again, and dictionaries rot." They do, and Chapter Two shows where they lived and why they drifted from the systems they described. The Type Contract differs in four ways. It covers boundary fields only, the fields that cross to another system, company or reader, not every column in the estate. It has seven fixed lines, short enough to fill in for one field in minutes. Each line has an executable test that runs at the boundary, so a contract that no longer matches its data fails loudly in the pipeline instead of rotting quietly in a document. And each code table it names has one accountable owner. Chapter Sixteen shows the test, not the document.

"Data contracts, schema registries, dbt tests and FHIR profiles already do this." They do part of it, and a company that has them should build on them. What they reliably carry is structure and syntax: field names, storage types, required and optional, allowed patterns, compatibility between versions. What they rarely carry, unless someone adds it by hand, is the currency of an amount, the time basis of a timestamp, the edition of a code system or what zero means. Chapter Sixteen maps the seven lines onto those tools, one by one, and adds only what each omits. The contract is a checklist for what those tools must say, not a rival to them.

"Models keep improving; give the agent more context and it will work out the rules." Context helps with rules that exist somewhere the model can reach. It cannot help with a fact that was never written. Fire panel 2 never recorded an offset. No inference, however good, recovers from the panel's log which pass of the repeated hour 01:42:10 belonged to; the answer came from a different clock entirely. What a good model can do is notice that 2 November 2025 was a fall-back night, see that a wall time inside the repeated

hour is ambiguous, and ask. The Type Contract should require that behaviour: when line 4 is missing for a value in an ambiguous window, the right output is "ambiguous, need offset", not a number of minutes.

OUR VIEW, AS OF NOVEMBER 2026

Better models will make more of the rules that exist and fewer silly mistakes with them. They will not make the missing rules exist. The cheapest place to supply a rule is where the value is born; the most expensive is in an agent's answer, after a person has acted on it.

ERA CARD

Reader. Every reader, from the clerk to the model, needed the rules from somewhere. Earlier readers rarely invented them; the model may.

Security. A value read under the wrong rules can be well-formed, authorised and harmful; line 7 makes who may receive it part of the value.

Cost. Minutes per boundary field to write a contract; hours per incident to reconstruct one afterwards, as the night shows.

AI. An agent needs the contract with the value, and an instruction to stop when a line is missing.

Monday rule: list the ten fields that cross your most important boundary, and for each one ask which of the seven lines a reader could find today without asking a person.

CHAPTER TWO

Where the Contract Lives



THE ERAS OF THE RECORD

The rules for reading the alarm time of 2 November 2025 did exist that night. The panel's configuration knew its clock ran on local time. The facilities vendor's technicians knew the export wrote wall time without an offset. The central monitoring station knew its own clock was synchronised. None of that knowledge was in the nineteen characters that reached the claim file. Every era of computing has kept the rules for its values somewhere: on the page, in a drawer, in a program library, in the database, in a schema file, in an API document. Many of the failures in this book happened at the moment a value moved to a reader who did not have that place. This chapter walks the places in order, briefly, because the history explains a decision every data leader now faces: **for each field that crosses a boundary, name where its contract lives, and make sure that place travels with the value or is reachable by every reader**.

On the page and in the drawer

In a handwritten ledger the rules sat a few centimetres from the value. A printed heading above a ruled column said what the figures were and, often, in what money. A clerk reading line 400 could look up and see. The heading did not travel when a figure was copied into a letter, but the page itself kept value and rule together, and the reader of the page always had both.

Punched cards separated them. A card held digits in columns, and nothing on the card's punched area said which columns were a date or where the decimal point fell. That knowledge lived on paper beside the deck. IBM's *Form and Card Design* manual of 1961 (form C20-8078) shows the practice. It describes a Multiple Card Layout form, "which should be used when planning several cards simultaneously or a new card that will be used with existing cards", and a Card Design Work Sheet for each field. Its rule for consistency is a rule about meaning: "A given field of information in the new card should be placed in the same columns previously assigned to it in the other cards." For amounts it advises: "Use

preprinted decimals and commas in boxes where amounts will be machine-interpreted." The point was printed for the human eye; the machine read only the digits, and the layout form said where the point belonged.

The same manual knew what happens when a reader lacks the rules. Reports printed on blank stock, it warns, "will probably be vague to outsiders, such as auditors, as well as to many insiders, unless the printer itself can be used to print column headings". The layout form worked for the people who had it. A deck carried to another department, without the form, was a box of numbers.

In the program and the dictionary

On tape and in the first databases the rules moved into code. The COBOL copybook, a shared record description included by every program that read the file, carried each field's position, length, type and the implied decimal point; *From Tally to Token* covers the copybook and packed decimal, and that story is not retold here. The copybook served its programs well. It said nothing about units or about what a code meant, and a reader outside the program library had no copy.

The data dictionary was the attempt to give the rules a home of their own: a register of every data element, its definition, format, owner and the systems that used it. In the United States the idea reached a national standard. ANSI X3.138-1988 defined an Information Resource Dictionary System, and the federal government adopted it for its own use as FIPS 156. On 29 July 1997 the National Institute of Standards and Technology announced that the Secretary of Commerce had approved the withdrawal of FIPS 156, along with 32 other federal standards. The design had a structural weakness that the book's first objection names: a dictionary is a separate system. Nothing forces a change in the data to be reflected in the dictionary, and nothing checks a value against its definition when the value moves.

In the database: the catalog

The relational database pulled the rules back next to the data. In two articles in *Computerworld*, on 14 and 21 October 1985, E. F. Codd set out twelve rules that a system must meet to call itself relational. The fourth concerns the catalog: the description of the database is held at the logical level just like ordinary data, so that authorised users can query it with the same relational language they use for the data itself. Codd restated the rule in his 1990 book on the second version of the relational model.

The catalog was a real advance. The database maintains it itself, so it describes the tables as they are. Any authorised reader can ask it which columns exist, their types, lengths, scale and whether they may be null. It is, in the terms of Chapter One, a good home for most of line 1 (type, precision and scale, though not the rounding rule) and part of line 6.

It is a poor home for the rest. A catalog says a column is DECIMAL(18,2); it does not say euros. It says TIMESTAMP; whether the values are instants or wall-clock readings, and in which zone, is outside its vocabulary. A code table can live in the database beside the data, but the link between a code column and its table is a convention unless someone declares it. And the catalog stays in the database. An extract, a CSV export or a nightly copy to another system takes the rows and leaves the catalog behind, which is how a perfectly described column becomes a bare number on the other side of a hop.

In the message: XML Schema, registries and API documents

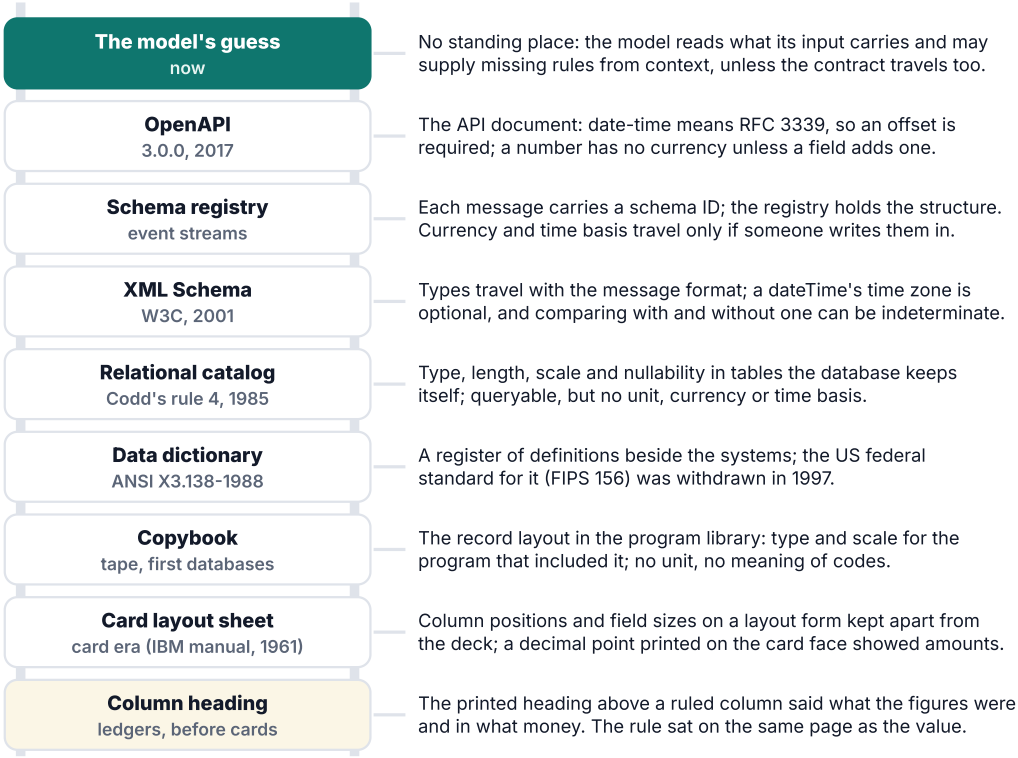
When data began to move between companies as messages, the rules moved into schema documents that travel alongside, or are referenced by, the message itself. The W3C published *XML Schema Part 2: Datatypes* as a Recommendation on 2 May 2001. It defines a decimal type that "represents arbitrary precision decimal numbers" and a dateTime type whose time zone is optional. It also confronts the night's problem directly. To compare a dateTime that has a time zone with one that does not, it treats the zoneless value as somewhere in a band from +14:00 to -14:00, and where the answer depends on the unknown zone, the result is neither earlier nor later but indeterminate. The rule is narrower than the night needed. Two zoneless values, like the claim file's, it compares field by field, exactly as the agent did. Twenty-four years before the agent compared 01:20 with 01:42, the standard had admitted that a missing zone can make an order undecidable, but only against a value that has one; two bare wall-clock readings it ranks without complaint.

Event streams brought the schema registry. Confluent's documentation describes its Schema Registry as providing "a centralized repository for managing and validating schemas for topic message data", for Avro, Protobuf and JSON Schema; with Confluent's serializers each message carries a short schema ID instead of the whole schema. Avro's own container files go one step further: a file carries its schema, and "all objects stored in the file must be written according to that schema". Here the rules finally travel with the value by design. What travels, though, is what the schema says: field names, types, precision and scale, which fields are optional. A currency, a time basis or a code-system version travels only if someone writes it into the schema, and the formats do not ask.

The API document did the same for request and response. The OpenAPI Specification, version 3.0.0 of 26 July 2017, lets an API declare that a string has the format date-time, defined by reference to RFC 3339. That reference carries real weight, because RFC 3339's grammar requires an offset in every date-time: a server that honours the format cannot send the panel's text. The same specification has a number type and no currency, so an amount's currency is a separate field or nothing.

FIGURE 2.1

Where the contract lived, place by place: each place served its own reader, and the value often moved without it



From the column heading to the API document, the rules that make a value usable sat in a different place in each era; the model has no standing place of its own and sees only what its input carries.

Author's diagram based on IBM C20-8078 (1961), FIPS 156 and its 1997 withdrawal notice, Codd (Computerworld, October 1985), W3C XML Schema Part 2 (2001), Confluent Schema Registry documentation and the OpenAPI Specification 3.0.0.

In the prompt: the model's guess

The newest reader has no standing place of its own. A language model receives only what the application around it supplies: a prompt, a tool's response, retrieved context, often text: a row rendered as words, a document extracted from a PDF, a tool's JSON response. If the catalog, the schema or the API document is not in that input, the model does not have it. In that it resembles the person handed a printed report without its layout form; the difference is in what each does next. It is not looking at a column typed TIMESTAMP; it is looking at the characters 01:42:10.

Earlier readers failed differently when the rules were missing. A program could fail to parse, or compute wrongly in the same way every time. A person could ask the colleague who kept the layout form. A model may do neither. Asked to compare two times, it compares them,

and supplies from context whatever rules the comparison needs. Often its guess is right, which is exactly what makes the wrong guesses hard to see. The claims-triage agent of the prologue did not have a catalog, a schema or a contract. It had two strings and a request to summarise.

The ladder in Figure 2.1 is not a story of progress from bad places to good ones. Each place served its own reader: the column heading served the clerk, the catalog served SQL, the registry served the stream's consumers. ISO/IEC 11179's framework names the limit they share: the meaning that names and definitions give "is limited, because names and definitions convey limited information about the object". The practical problem today is that one value now passes through several of these places in a single day, and the last reader, often a model, sees only what the previous hop chose to pass on. A Type Contract does not replace the catalog, the schema or the API document. It is the checklist that says what each of them must carry for the field to survive the next hop, and a test that fails when it does not.

WHAT HASN'T CHANGED

From the 1961 layout form to the 2017 API document, the rules worked for the readers who had the document and failed for the readers who did not. The question has always been the same: who reads this value next, and will they have the place where its rules are kept?

ERA CARD

Reader. Each era's reader had its own place for the rules: heading, layout form, copybook, catalog, schema, API document; the model has only what its input carries.

Security. A schema that validates structure but not meaning lets a well-formed value through to any reader the transport allows.

Cost. A rule written into the place the next reader uses costs little; a rule left in the previous system's catalog is paid for at every hop.

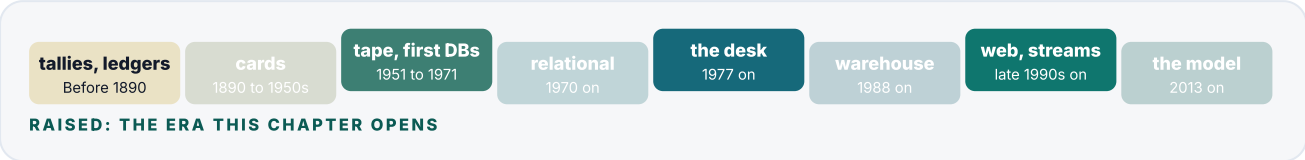
AI. Give the model the contract as part of the data it reads, or it will reconstruct one from context.

Monday rule: for one dataset that leaves your company or reaches an agent, write down where each field's rules live today, and whether that place reaches the last reader.

PART II · KINDS OF VALUE

CHAPTER THREE

Integers and Floats



On 1 May 2015 the US Federal Aviation Administration published an airworthiness directive, AD 2015-09-07, for every Boeing 787 in service. A 787 that "has been powered continuously for 248 days can lose all alternating current (AC) electrical power", the directive said, because "a software counter internal to the GCUs", the generator control units, "will overflow after 248 days of continuous power". Nothing in the counter was wrong for the first 247 days. The number simply reached the end of the space it had been given. Part II of this book takes the kinds of value one at a time, and numbers come first because every other kind is eventually stored as one. The decision this chapter changes: **every numeric field that crosses a boundary states its range, what happens at the edge, and whether it is exact**.

A width is a promise about range

How a machine holds an integer in bits, and how it adds them, is the subject of the Every Pixel Matters book *From Bit to Pixel*; here one fact is enough. An integer type has a **width**, a fixed number of bits, and the width sets the range. Sixty-four bits in the usual signed form, called two's complement, hold every whole number from minus 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Thirty-two bits stop at 2,147,483,647. Figure 3.1 lists the common widths.

FIGURE 3.1

Integer widths and limits: the same width has different names and different behaviour at the edge

Width	Signed range	Unsigned range	Where you meet it
8 bits	-128 to 127	0 to 255	Java byte; Parquet INT(8, true or false)
16 bits	-32,768 to 32,767	0 to 65,535	Java short; PostgreSQL smallint; Parquet INT(16)
32 bits	-2,147,483,648 to 2,147,483,647	0 to 4,294,967,295	Java int; PostgreSQL integer; Parquet INT(32)
64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0 to 18,446,744,073,709,551,615	Java long; PostgreSQL bigint; Parquet INT(64)
binary64 double: safe-integer range (every integer up to 2^53 is exact; above it, not all)	-(2^53 - 1) to 2^53 - 1, that is 9,007,199,254,740,991	not applicable	ECMAScript Number; JSON numbers read by JavaScript; RFC 8259's interoperable range

The ranges are the arithmetic of two's complement and of a 53-bit significand; one past the limit, Java wraps with no signal, PostgreSQL refuses the store and a binary64 conversion takes the nearest double, so the behaviour is set by the language or engine, not by the width.

Source: Java Language Specification SE 21, sections 4.2.1 and 4.2.2; PostgreSQL documentation, 8.1 Numeric Types; Apache Parquet LogicalTypes (INT); ECMA-262, Number.MAX_SAFE_INTEGER; RFC 8259 section 6; unsigned ranges computed by the author.

The width is not the whole contract. What happens one step past the limit is decided by the language or the engine, and they disagree. The Java Language Specification, section 4.2.2, is blunt: "The integer operators do not indicate overflow or underflow in any way." Section 15.18.2 says what happens instead: an overflowing addition returns "the low-order bits of the mathematical sum", so one more than the largest `int` is the most negative `int`, with no error. PostgreSQL takes the opposite view for its `smallint`, `integer` and `bigint`

columns: "Attempts to store values outside of the allowed range will result in an error." A counter that wraps silently in the application and fails loudly in the database is one value with two contracts, and the two meet at every write.

Signedness is part of the type too. The Parquet format lets a writer declare an integer column as `INT(32, false)`, unsigned, which holds up to 4,294,967,295. Loaded into a signed 32-bit column, every value above 2,147,483,647 either fails or turns negative, depending on the reader. Nothing in the digits says which width the writer meant. The declaration does, if it travels.

> **SOURCED FAILURE: THE 787 GENERATOR CONTROL UNITS, 2015**
>
> Airworthiness Directive 2015-09-07 (Amendment 39-18153, Docket FAA-2015-0936, 80 FR 24789) applied to all Model 787 airplanes and took effect on 1 May 2015, the day it was published. The FAA wrote that it had "been advised by Boeing of an issue identified during laboratory testing": after 248 days of continuous power a counter inside the generator control units would overflow, the units could go into failsafe mode at the same time, and the airplane could lose all AC electrical power, "which could result in loss of control of the airplane". The required action was operational, not a code change: an electrical power deactivation, repeated "at intervals not to exceed 120 days". A later directive, AD 2018-20-15 (83 FR 52131, 16 October 2018), superseded it and required new GCU software. The directive does not describe the counter's width or unit, and neither does this book. Its lesson for data is the one in its text: the limit was identified in laboratory testing, and the interim remedy was a power cycle that keeps every airplane short of it.

IEEE 754: the standard for approximate numbers

For numbers with fractions, or too large for an integer, nearly every processor and language uses **floating point**: a sign, a significand (the digits) and an exponent that moves the point. The rules come from one standard. IEEE Std 754-1985, the IEEE Standard for Binary Floating-Point Arithmetic, was approved on 21 March 1985. Its 2008 revision, approved on 12 June 2008 and retitled the IEEE Standard for Floating-Point Arithmetic, added decimal formats, which Chapter Four takes up. IEEE 754-2019, approved on 13 June 2019, took the 2008 text as its base document; the IEEE committee that maintains it says it is "now in maintenance mode, and will expire in 2029". Each edition has an international twin: IEC 60559:1989 for the 1985 text, ISO/IEC/IEEE 60559:2011 for 2008, and ISO/IEC 60559:2020 for the 2019 revision.

The format most data meets is **binary64**, the "double". It has a 53-bit significand. Java's `double` values "exactly correspond to" it (JLS section 4.2). ECMAScript, the standard behind JavaScript, defines its Number type as the binary64 values: "exactly

18,437,736,874,454,810,627 (that is, 2^64 - 2^53 + 3) values", with all the not-a-number patterns folded into one NaN. That count is the useful way to think about a double. It is a finite set of exact numbers, chosen by a rule, and every arithmetic result is replaced by the nearest member of the set.

Why a decimal fraction is not exact in binary

A decimal fraction such as 0.1 is one tenth. In base 10 it needs one digit. In base 2 it never ends: 0.0001100110011... with the block 0011 repeating, exactly as one third in decimal is 0.333... without end. A binary format with 53 significant bits must stop somewhere, so it stores the nearest number it can hold. Figure 3.2 shows the three doubles around 0.1, computed for this book. The one chosen for 0.1 is larger than one tenth by about 0.00000000000000056.

FIGURE 3.2

A binary64 holds exact values on a grid: near 0.1 the steps are tiny, near 2^53 they are 2

What a binary64 (IEEE 754 double) can hold near 0.1 and near 2^53 (9,007,199,254,740,992)

Neighbour below 0.1 binary64	0.09999999999999999916733273...	Exact value of the double just under the one for 0.1
What "0.1" becomes nearest double	0.10000000000000000055511151...	Closest of the two; the gap between doubles here is 2^-56
Neighbour above 0.1 binary64	0.10000000000000000194289029...	No double lies between these three values
2^53 - 1 Number.MAX_SAFE_INTEGER	9007199254740991	Exact; the largest n for which n and n + 1 are both exact
2^53 binary64	9007199254740992	Exact; from here the gap between doubles is 2
2^53 + 1, as typed read into binary64	9007199254740993 → ...992	Not on the grid: silently stored as its neighbour
2^63 - 1, BIGINT max read into binary64	9223372036854775807 → ...808	Rounds to 2^63, one past the column's own maximum

Every double is an exact number; a decimal fraction or a large integer that is not on the grid is replaced by the nearest one that is, without an error.

*Author's computation, 20 November 2026: Python 3.13.3 (math.nextafter, math.ulp, decimal.Decimal of each double); Node.js 22.20.0 for 2**53 + 1; ECMA-262, Number.MAX_SAFE_INTEGER.*

"Floats are inaccurate" is the wrong lesson from this, and it leads to the wrong fix. Every double is an exact number, and IEEE 754 defines precisely which one an operation returns. What is lost is the decimal value the writer meant, at the moment it is converted. *From Tally*

to Token showed the consequence for money: the sum 0.1 + 0.2 and a one-cent check that fails. Here the point is the grid. Near 0.1 the gap between neighbouring doubles is 2^-56, about 0.000000000000000014. Near 2^53 it is 2. Above 2^53, about 9 quadrillion, a double cannot hold an odd number at all, and the bottom half of Figure 3.2 shows what happens to one: 9,007,199,254,740,993 is stored as 9,007,199,254,740,992, without an error. The largest 64-bit integer, a common maximum for a `bigint` key, rounds up to 2^63, which is one more than the column it came from can hold. A reader that turns such a key into a double and writes it back has made a value its own database rejects.

The useful questions for a contract follow. Does the field hold measurements, where a relative error far below the instrument's is harmless? Then binary64 is fine, and the contract states the tolerance. Does it hold counts, identifiers or money, where the business rule is exact? Then it needs an integer of declared width or a decimal type, and the contract forbids the double.

SOURCED FAILURE: THE PATRIOT BATTERY AT DHAHRAN, 25 FEBRUARY 1991

The General Accounting Office's report GAO/IMTEC-92-26, *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia* (4 February 1992), is the record. On 25 February 1991 a Patriot battery at Dhahran "failed to track and intercept an incoming Scud", which "hit an Army barracks, killing 28 Americans" (p. 1). The weapons control computer was "based on a 1970s design with relatively limited capability to perform high precision calculations" (p. 4), older than IEEE 754. Time was "kept continuously by the system's internal clock in tenths of seconds but is expressed as an integer"; to predict where the Scud would appear, time had to be converted to a real number, and because the registers "are only 24 bits long, the conversion of time from an integer to a real number cannot be any more precise than 24 bits" (pp. 5 to 6). The error grew with running time. Israeli data received on 11 February showed a 20 percent shift of the radar's range gate after 8 hours; the Army estimated that the range gate would shift 50 percent after about 20 hours, a shift at which the system no longer tracks the target (pp. 7 to 8). Alpha Battery had been running "for over 100 consecutive hours" (p. 9). Appendix II gives the inaccuracy after 100 hours as 0.3433 seconds and the range gate shift as about 687 meters. Software that compensated was released on 16 February and reached Dhahran on 26 February, the day after the attack.

The report does not show the arithmetic, and it is worth doing because it matches. One tenth, cut off after 23 binary places, is short by 0.000000095367431640625. Multiply by the 3,600,000 tenths in 100 hours and the clock is behind by 0.3433 seconds; by the 288,000 tenths in 8 hours, 0.0275 seconds. Both are the figures in GAO's Appendix II. Each tick was nearly perfect. The integer counter was exact. The loss was the conversion of one decimal fraction into a fixed number of binary digits, multiplied by a run time no one had written into the contract.

The language decides: JavaScript's Number and BigInt

A value's type is often chosen not by its writer but by the language of its reader. JavaScript is the common case. Its ordinary numbers are binary64, so integers are safe only up to a stated limit; ECMA-262 says "The value of Number.MAX_SAFE_INTEGER is 9007199254740991 (2^53 - 1)", the largest integer n for which n and n + 1 are both exact. In Node.js 22, `2**53 + 1 === 2**53` is true. A 64-bit account number, order number or tracking id handed to a JavaScript reader as a number can change its last digits without a sound.

JavaScript is not limited to that one numeric type. The 11th edition of ECMA-262, ECMAScript 2020 (June 2020), introduced "BigInt, a new number primitive for working with arbitrary precision integers"; the specification says a BigInt "may be any size and is not limited to a particular bit-width". The catch is at the boundary. The current specification's own JSON serializer refuses it: "If value is a BigInt, throw a TypeError exception." So a program that does the right thing inside, with BigInt, still has to decide how the value leaves, usually as a string, and the reader on the other side has to know that the string is an integer.

JSON itself leaves the question open. RFC 8259, section 6, "allows implementations to set limits on the range and precision of numbers accepted" and says only that integers in the range minus (2^53 minus 1) to 2^53 minus 1 "are interoperable". The grammar has no width, no signedness and no scale, so the same text can give two readers two values. Chapter Eleven shows two standard parsers doing exactly that.

TYPE CONTRACT: INTEGERS AND FLOATS

1. Type, precision and rounding. For an integer: width, signed or unsigned, and the behaviour past the limit (an error, never a silent wrap). For a float: binary64 or binary32, and the tolerance the use can accept; exact quantities are not floats. *Test:* the boundary suite sends the maximum and the maximum plus one; the second must be rejected, not stored.

2. Encoding and serialization. For binary files, byte order and width; in JSON, integers beyond 2^53 minus 1 travel as strings, and floats as decimal text that round-trips. *Test:* 9007199254740993 and 0.1 pass through the interface and come back as the same text.

3. Unit or currency. What is counted or measured, and in which unit (Chapter Eight); a bare number is not accepted. *Test:* a numeric field without a declared unit or a declared "count of" is held.

4. Time basis. For a counter: what it counts from (power on, an epoch), its tick (tenths of seconds, milliseconds) and when it resets; the longest run the system must survive. *Test:* the counter is driven past its limit in a test environment before

production reaches it.

5. Identity scope. An integer identifier is a name, not a quantity (Chapter Seven): no arithmetic, no conversion to a float. *Test:* a schema change that turns an identifier column into a floating type fails review.

6. Absence. Zero is a value; NaN is not a marker for missing (Chapter Ten). *Test:* null, 0 and NaN arrive as three different things and are kept apart.

7. Authority. A named owner for the range decision and for widening a column; which system's limit wins when two disagree. *Test:* any change of width, signedness or numeric type is approved by that owner before deployment.

OUR VIEW, AS OF NOVEMBER 2026

Integer overflow and binary rounding are among the oldest known hazards in computing, and they keep appearing because the limit lives in a type declaration that the next reader never sees. The cheapest control we know is the boundary test of line 1: send the maximum plus one and watch what happens.

ERA CARD

Reader. Embedded controllers, databases, Java and JavaScript runtimes and JSON parsers; each applies its own width and its own rule at the edge.

Security. A wrapped integer is a classic attack surface: a length or a count that turns negative or small can bypass a check written for the large value it was.

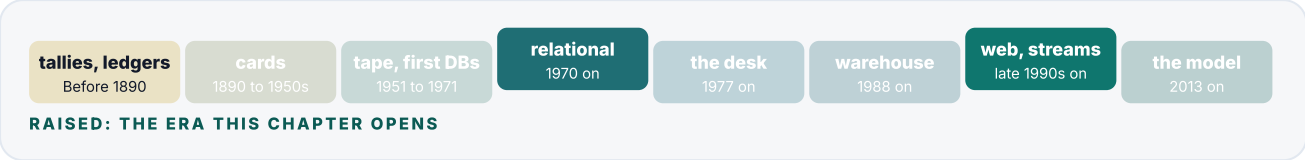
Cost. The 787 directive imposed a power cycle at least every 120 days on every airplane until new software arrived; a limit found late is paid in operations.

AI. A model reading "9007199254740992" cannot know it was once ...993; the precision has to be preserved before the value reaches any reader, human or model.

Monday rule: list the integer identifiers and counters your systems pass as JSON numbers or load into floating types, and move every one that can exceed 2^53 minus 1 to a string or an integer of declared width.

CHAPTER FOUR

Money and Decimals



The guest had booked three nights on the hotel group's EU website at EUR 411.52 a night. After the fire on Sunday 2 November 2025 the hotel refunded the whole booking, all three nights: EUR 1,234.56. The group keeps its books in US dollars, so someone had to convert the refund, and the obvious source, the European Central Bank's reference rate, is not published on a Sunday. Two teams converted it, each in a way that looked correct, and got two amounts. Neither was a mistake in arithmetic. Each team had filled in a rule that nobody had written: which rate, which day, where to round and how. The decision this chapter changes: **an amount travels with its currency, its scale, its rounding rule and the point where rounding happens, and a converted amount also carries its rate, the rate's date and its source**.

A decimal type says how many digits and where the point is

Chapter Three ended with a rule: exact quantities are not floats. Money is the main exact quantity in most companies, and SQL's answer is the **decimal** type, written `DECIMAL(p, s)` or `NUMERIC(p, s)`. PostgreSQL's documentation gives the two numbers their plain meaning. The **precision** is "the total count of significant digits in the whole number, that is, the number of digits to both sides of the decimal point"; the **scale** is "the count of decimal digits in the fractional part". A column declared `NUMERIC(3, 1)` holds minus 99.9 to 99.9.

The declaration also says what happens to a value that does not fit, and here products differ in ways a migration feels. In PostgreSQL, extra fractional digits are rounded to the scale, and too many integer digits raise an error. In SQL Server, `decimal` and `numeric` are synonyms, precision runs from 1 to 38, and "the default precision is 18" with a default scale of 0. Conversion to a lower scale rounds by default; a value too large for the precision raises arithmetic overflow error 8115 when `ARITHABORT` or `ANSI_WARNINGS` is on, and becomes NULL when both are off, with a warning unless `ARITHIGNORE` is on. One consequence follows from the documentation: a column declared as plain `decimal`, with no arguments,

is `decimal(18, 0)`, and the refund of EUR 1,234.56 stored there becomes 1235. In Db2 the maximum decimal precision is 31 digits. A `DECIMAL(38, 4)` column from one engine has no exact home in the other.

Files carry the same idea. *From Tally to Token* showed how Parquet stores a decimal as an integer with a scale; the specification allows a 32-bit integer for precision up to 9 and a 64-bit integer up to 18, and asks writers to put precision and scale into the file's schema, so the scale travels with every column. What does not travel is the currency. `DECIMAL(18, 2)` says "two places". It does not say "euros", and it does not say whether two places is right for the currency in the row.

Decimal floating point

The 2008 revision of IEEE 754 brought decimal arithmetic into the floating-point standard: formats that store decimal digits rather than binary ones, with up to 7, 16 or 34 digits of precision. Mike Cowlishaw, whose General Decimal Arithmetic specification describes the model, notes that this arithmetic "is now included in the IEEE 754-2008 and the ISO/IEC/IEEE 60559:2011 standards", and the current edition's abstract covers "binary and decimal floating-point arithmetic". Db2 exposes it as `DECFLOAT`, with a precision of 16 or 34 digits.

Python's `decimal` module shows what the change buys: "In decimal floating point, 0.1 + 0.1 + 0.1 - 0.3 is exactly equal to zero." Decimal fractions within the precision are held exactly, so the binary surprise of Chapter Three disappears. Two things remain. A decimal float is still floating: when a result has more digits than the precision, it is rounded, under a rule set by a context, and Python's default context uses 28 digits and rounds half to even. And a decimal float has no fixed scale, so 1234.56 and 1234.560 can both appear. For a ledger, a fixed scale per currency is still the contract; decimal floating point is a good engine for the arithmetic in between.

A currency is part of the value

An amount without its currency is half a value. The international list is **ISO 4217**, maintained by SIX Financial Information as maintenance agency on behalf of ISO. Each currency has a three-letter code (EUR, USD), a three-digit number (978, 840) and a **minor unit**: the number of decimal places used with it. The list SIX published on 17 September 2026 gives most currencies two, but not all. Figure 4.1 counts them.

FIGURE 4.1

Minor units by currency: the same integer of minor units is four different amounts, or none

Minor unit	Codes in List One	Examples	123456 minor units means
0	17	JPY, KRW, CLP, ISK, VND	JPY 123,456
2	139	EUR, USD, GBP, TRY	EUR 1,234.56
3	7	BHD, IQD, JOD, KWD, LYD, OMR, TND (all of them)	KWD 123.456
4	2	CLF (Unidad de Fomento), UYW (Unidad Previsional); both marked as funds	CLF 12.3456
N.A. (none defined)	13	XAU (gold), XAG (silver), XDR, XTS (testing), XXX (no currency)	nothing: no minor unit is defined

Counts are distinct alphabetic codes in ISO 4217 List One as published by SIX on 17 September 2026; a field that stores minor units must carry the code, or the exponent is a guess.

Source: SIX, ISO 4217 Currency Code List One (published 2026-09-17), parsed by the author on 20 November 2026.

Seventeen codes have a minor unit of 0, no decimal places, among them the Japanese yen. Seven have three, the Kuwaiti and Bahraini dinars among them. Two have four: Chile's Unidad de Fomento (CLF) and Uruguay's Unidad Previsional (UYW), both marked in the list as funds rather than ordinary currencies. Precious metals and special codes such as XXX are marked N.A.: no minor unit is defined at all, which is not the same as zero. A system that stores amounts as integers of the minor unit, a sound choice, must carry the code or the exponent with every amount, because 123456 is EUR 1,234.56, JPY 123,456 or KWD 123.456. A column fixed at two places rounds away a dinar's third decimal and invents cents for a yen.

The list is also a dated document: SIX republishes it as it changes, and the edition used here is the one of 17 September 2026. A contract names the list it relies on and its publication date, as it would name the edition of a code system (Chapter Eight).

Rounding modes, and who decides

When a result has more digits than the scale, a **rounding mode** picks one of the two neighbours. The half modes agree on every result except a tie, where the discarded part is exactly half. Java's `RoundingMode` documentation states the two that matter for money. **Half-up** rounds a tie away from zero, "the rounding mode commonly taught at school"; 2.5 becomes 3 and minus 2.5 becomes minus 3. **Half-even** rounds a tie to the even neighbour, so 2.5 becomes 2; it "statistically minimizes cumulative error when applied repeatedly", and "is sometimes known as 'Banker's rounding,' and is chiefly used in the USA".

Defaults hide the choice. Python's decimal context rounds half to even unless told otherwise. PostgreSQL, in one database, does both: "the numeric type rounds ties away from zero, while (on most machines) the real and double precision types round ties to the nearest even number", so `round(2.5)` is 3 as numeric and 2 as double precision. A developer who never chose a mode has still shipped one.

The choice belongs to the owner of the financial record, usually finance, sometimes a regulator, and the best model of how to write it down is a regulation about the euro. When the euro was introduced, Council Regulation (EC) No 1103/97 of 17 June 1997 set the arithmetic in two articles. Article 4: the conversion rates "shall be adopted with six significant figures"; they "shall not be rounded or truncated when making conversions"; "Inverse rates derived from the conversion rates shall not be used"; and an amount moving from one national currency to another goes through the euro, with the euro amount, which "may be rounded to not less than three decimals". Article 5: amounts are "rounded up or down to the nearest cent", and "If the application of the conversion rate gives a result which is exactly half-way, the sum shall be rounded up." Article 4 also closes the door on clever alternatives: "No alternative method of calculation may be used unless it produces the same results."

The regulation governed conversions between the euro and the national currency units it replaced. It does not govern euro to dollar conversion, and nothing in this chapter says it does. Its value here is its form. In a few sentences it named the precision of the rate, forbade the inverse, fixed the intermediate step and its precision, said where rounding happens and settled the tie. That is line 1 of a Type Contract for money, written by a legislator.

A rate is a value with a date and a source

An exchange rate looks like a constant and behaves like a measurement: it has a time and an author. The ECB publishes euro reference rates against 29 currencies (30 on the night of the fire; the Bulgarian lev's last reference rate is dated 31 December 2025), and its page says when: "The reference rates are usually updated at around 16:00 CET every working day, except on TARGET closing days." It also says what they are for: they are "published for information purposes only", and "Using the rates for transaction purposes is strongly discouraged." A company that books conversions at ECB rates has made a policy choice, which is allowed; the contract should say so, and say which series it uses.

For the night of the fire the ECB's data portal, series EXR.D.USD.EUR.SP00.A, has these observations: 1.1550 US dollars per euro on Thursday 30 October 2025, 1.1554 on Friday 31 October, nothing on Saturday 1 or Sunday 2 November, and 1.1514 on Monday 3 November. "The rate for 2 November" does not exist. A system asked for it returns an error, a null, Friday's value or Monday's, depending on how it was written, and only the last two look like answers.

THE HOP · 6 · 2025 · EU BOOKING ENGINE (EUR) TO GROUP FINANCE (USD), THEN THE PAYMENT API (JSON)

Record	Room refund (One Night, Every Value, illustrative), hop 6, 2025
From	EU booking engine (EUR)
To	Group finance (USD), then the payment API (JSON)
Mechanism and schedule	The refund. The group refunds the whole three-night booking after the fire; finance converts each night at the last published reference rate and rounds each line half-up; the payments team converts the total once
Format change; what was lost or assumed	PRECISION TIME 3 x EUR 411.52 = EUR 1,234.56. Finance converts each night at Friday 31 October's ECB rate (1.1554) and rounds half-up: USD 1,426.41. Payments converts the total at Monday 3 November's rate (1.1514): USD 1,421.47, a cent from the per-line figure at that rate. The two records disagree by USD 4.94. **No reference rate is published on a weekend, so the rate date is an unstated choice (the contract's time-basis line should say: last published rate before the transaction date); the rounding point and rule are unwritten**
Accountable owner	Group finance
Who can change it	Finance analysts; the payments service
Classification; retention or erasure duty	Financial record
Read by (people, apps, models, agents)	Finance, the payment provider, auditors
Cost per month; basis	One-off: a reconciliation break reviewed by finance (not monthly) (illustrative)

Hop 6 is that refund. The booking engine held three lines of EUR 411.52, exact decimals with the currency beside them. Group finance converted each night at the last published reference rate, Friday's 1.1554, and rounded each line half-up: 411.52 times 1.1554 is 475.470208, so each line became USD 475.47, and the three lines USD 1,426.41. The payments service, which sends the refund through the payment API as JSON, converted the total once, on Monday afternoon, after the ECB had published that day's rate, 1.1514: 1,234.56 times 1.1514 is 1,421.472384, or USD 1,421.47. The two records of one refund differed by USD 4.94, and the break went to finance to review. No step was wrong by its own rules. The rules were unwritten: the rate date, the rounding point and the rounding mode each lived in a different team's code.

Figure 4.2 separates the effects, with every value computed for this book. The rate date moved the refund by dollars. The rounding point moved it by a cent, but only on some days: at Friday's rate, per-line and total conversion both give 1,426.41, because the three discarded remainders, 0.000208 each, add up to 0.000624, too small to cross a cent; at Monday's rate, per-line conversion gives 1,421.46 and the total gives 1,421.47. A test written on Friday's rate would have passed. The rounding mode, half-up or half-even, changed nothing here, because none of these products ends in an exact half; it matters on the rare tie, while the rounding point can change the cents on any amount, as Monday's rate shows.

FIGURE 4.2

One refund, three places where digits change: the rate date moves the dollars, the rounding point at most a cent, and the binary sum nothing visible this time

Hop 6, the room refund: 3 x EUR 411.52 = EUR 1,234.56, converted to US dollars

Booking engine EUR, decimal, scale 2	`3 x 411.52 = 1234.56`	Exact in decimal; the currency code travels beside it
Rate: ECB reference Friday 31 October 2025	`1 EUR = 1.1554 USD`	Last published before Sunday 2 November; none on 1 or 2 Nov
Point 1: each night half-up, Friday rate	`475.470208 → 475.47; x 3 = 1426.41`	Finance's booking: three rounded lines, summed
Point 2: total once Friday rate	`1426.410624 → 1426.41`	Same cents at this rate: the digits happen to agree
Point 2: total once Monday 3 Nov, 1.1514	`1421.472384 → 1421.47`	Payments took the rate of the day it ran: USD 4.94 apart
Point 1 at Monday rate for comparison	`473.824128 → 473.82; x 3 = 1421.46`	Same rate, other rounding point: one cent apart
Point 3: 475.47 × 3 as JSON numbers, binary64 sum	`1426.41000000000008185...`	Prints as 1426.41; what is stored is the nearest double

With Friday's rate the per-line and total methods agree to the cent; with Monday's rate they differ by one cent, and the two rate dates differ by USD 4.94.

Author's computation, 20 November 2026, Python 3.13.3 decimal (ROUND_HALF_UP) and float; rates from the ECB Data Portal, series EXR.D.USD.EUR.SP00.A. Refund amounts illustrative (One Night, Every Value).

A third place where digits can change is downstream, and it is not a rounding rule at all: the binary sum. The payment API carried amounts as JSON numbers, and a reporting job summed finance's three line amounts, 475.47 each, in binary64. For these values the sum prints as 1426.41; the number actually held is 1426.41000000000008185452315956354141235351562 5, the nearest double. Nothing visible went wrong this time, which is the trouble with it: the decimal guarantee ended at the

JSON boundary, and whether the next sum comes out right is a matter of which digits happen to arrive. Chapter Eleven gives the format's side of this; the contract's side is to send amounts as strings or as integers of the minor unit, with the currency code.

TYPE CONTRACT: MONEY (THE ROOM REFUND OF HOP 6)

1. Type, precision and rounding. Decimal, precision 18, scale equal to the currency's minor unit (2 for EUR and USD); converted amounts rounded half-up to the minor unit once, on the converted total; any difference from the sum of converted lines posted to a named rounding account. *Test:* a value with more decimals than its currency's minor unit, or a binary floating type, is held; the boundary suite includes the three refund lines at both 1.1554 and 1.1514.

2. Encoding and serialization. In JSON, the amount as a string matching a decimal pattern, or as an integer of minor units, never as a JSON number; in tables, `DECIMAL(18, 2)` with the scale declared. *Test:* the API schema rejects a numeric type for amount fields, a house rule chosen because common JSON readers parse numbers as binary64.

3. Unit or currency. ISO 4217 alphabetic code beside every amount; a converted amount carries both codes, source and target. *Test:* the code exists in the List One edition named in line 7, and the amount's scale matches its minor unit.

4. Time basis. The transaction date is the refund date, 2 November 2025, in the ledger's zone. The rate is the last published rate before the transaction date: for this refund, 31 October 2025, 1.1554. A same-day rate is excluded on purpose, because the ECB publishes at about 16:00 CET, after many transactions of the day. The rate's date is stored with the converted amount. *Test:* a conversion without a rate date, or with a rate dated on or after the transaction date, is held; a Sunday transaction resolves to the previous working day's rate.

5. Identity scope. One refund of one booking, made of three night lines; the converted total references the lines and the rate record it came from. *Test:* every converted amount resolves to its source amount, its rate and its rule version.

6. Absence. A missing amount is not zero; a zero refund is written as 0.00 with a reason. A missing rate stops the conversion; it is never filled with the nearest available one by default. *Test:* an empty amount, or a conversion with no rate, is held, not posted.

7. Authority. Group finance owns the rounding rule and the rate policy; the rate source is the ECB reference rate, series EXR.D.USD.EUR.SP00.A, used as an accounting rate by company policy; ISO 4217 List One, the edition in force on the

transaction date (this book checks against the publication of 17 September 2026). The payment provider receives the amount, currency and reference, nothing more. *Test:* each conversion is stamped with the rule version and the rate source, and a change to either needs finance's approval.

Under this contract the refund is USD 1,426.41: Friday's rate, as finance chose, rounded once on the total, as payments did. At this rate the cents match finance's figure, but neither team followed the whole rule, and finance's per-line rounding changes to the contract's single point.

OUR VIEW, AS OF NOVEMBER 2026

Many money defects are not arithmetic errors. They are two correct calculations under two unwritten rules. The fix is a short, owned document of the kind Regulation 1103/97 is, with the rate date and the rounding point in it, enforced by tests that use more than one day's rate.

ERA CARD

Reader. Booking engines, ledgers, payment APIs, reporting jobs and auditors; each applies a scale, a mode and a rate date, written or not.

Security. Rounding is a control point: a rule nobody owns can be changed in code without review, and small differences repeated over many transactions are hard to see in any one of them.

Cost. A break of a cent or of a few dollars costs the same review time; the cost is the reconciliation, not the amount.

AI. An agent asked "what was the refund in dollars" will pick a rate if none is attached; the rate, its date and its source must arrive with the amount, or the agent must be told to ask.

Monday rule: ask finance to write, on one page, the rate source, the rate date rule, the rounding point and the rounding mode for every currency conversion, and add one test that runs the same conversion on two different days' rates.

CHAPTER FIVE

Text



The guest booked her room on the group's EU website and typed her name as she writes it: Işıl Yılmaz. By the time the insurer's adjuster opened the claim file in November 2025, the same woman appeared in four forms. The booking had her own spelling. The hotel's property system export said I??l Y?lmaz. The hospital, whose registration desk scanned her passport and took the name from its machine readable zone, had the passport's form, YILMAZ<<ISIL. A later file from the booking side, opened with the wrong setting, showed a string of accented capitals that nobody could read aloud. Every system had done what it was built to do, and the claim file could not match one person to herself. Text looks like the simplest kind of value, because people read it without effort. A machine reads bytes, and bytes become letters only under rules the bytes do not carry. The decision this chapter changes: **every text field that crosses a boundary declares its encoding, its normalization form and its length unit, and a conversion that cannot keep a character stops instead of guessing**.

Characters, code points, bytes and glyphs

Four different things hide behind the word "character", and most text failures come from treating two of them as one. The Unicode Standard draws the first line itself. Its section 2.2, "Characters, Not Glyphs", says that "characters are the abstract representations of the smallest components of written language that have semantic value", while glyphs "represent the shapes that characters can have when they are rendered or displayed". The letter ş is a character. The curve and the small hook a screen draws for it are a glyph, chosen by a font. How pixels and fonts turn characters into shapes is the subject of *From Code to Glass*, Book 2 of *Every Pixel Matters*; this chapter stays with what is stored.

Between the character and the bytes sit two more layers. A **code point** is the number Unicode assigns to a character, written U+ and four to six hexadecimal digits: ş is U+015F, LATIN SMALL LETTER S WITH CEDILLA, and the dotless ı is U+0131. The codespace runs from 0 to 10FFFF hexadecimal, 1,114,112 code points (section 2.4). An **encoding form**

turns code points into code units, and an encoding scheme turns those into a sequence of bytes. The same code point is one byte, two bytes or four bytes depending on the scheme, so the byte count of a name says nothing about its letter count until the encoding is known.

The current version is Unicode 18.0.0, published by the Unicode Consortium on 16 September 2026; it added 13,007 characters, for a total of 172,808. A company does not need to track every addition, but it should know which version its databases, search engines and validation libraries implement, because a character assigned in a newer version is unknown to an older library. How ASCII, EBCDIC, Unicode and UTF-8 came to be is told in *From Tally to Token*; here we start from where that history left off.

UTF-8, UTF-16 and the byte order mark

Two encodings carry most enterprise text today. **UTF-8** writes each code point as one to four bytes (definition D92 in chapter 3 of the standard). The ASCII letters keep their single byte, and every other character takes two or more. **UTF-16** writes each code point as one or two 16-bit code units; characters from U+10000 upward need a pair of code units called a surrogate pair (D91). Any environment whose strings count UTF-16 code units therefore counts such a character as two.

A 16-bit code unit is two bytes, and two bytes can be stored in either order. Big-endian puts the high byte first; little-endian puts it last. The **byte order mark** (BOM) is the code point U+FEFF placed at the start of a stream: written as FE FF it says big-endian, as FF FE it says little-endian. Table 2-4 of the standard allows a BOM in the UTF-16 encoding scheme and not in UTF-16BE or UTF-16LE, whose names already state the order. Without a BOM and without a higher-level protocol, definition D98 makes UTF-16 big-endian. Microsoft's code page list shows how easily that default is missed: its identifier 1200, named "utf-16", is "Unicode UTF-16, little endian byte order".

UTF-8 has no byte order question, yet a BOM still appears in UTF-8 files as the three bytes EF BB BF. What the standard says about it has changed, and teams quoting it from memory often quote the old text. Version 15.0 said: "Use of a BOM is neither required nor recommended for UTF-8." From version 16.0 the same section reads: "Use of a BOM is not required for UTF-8", and adds that "the Unicode Standard does not recommend for or against use of a BOM in UTF-8 data". Section 23.8 now gives practical rules, among them "If consuming UTF-8, recognize and discard a BOM", and warns that concatenating files without removing it inserts a stray U+FEFF character into the text. The same section notes that, read as Windows code page 1252, the three bytes appear as "ï»¿": the first characters of many corrupted CSV headers. For a Type Contract the rule is simple: say whether the producer writes a BOM, and make the reader accept both.

```
"Işıl Yılmaz", 11 code points
UTF-8      49 C5 9F C4 B1 6C 20 59 C4 B1 6C 6D 61 7A   (14 bytes)
UTF-16BE   00 49 01 5F 01 31 00 6C 00 20 00 59 01 31
           00 6C 00 6D 00 61 00 7A                     (22 bytes)
UTF-16LE   49 00 5F 01 31 01 6C 00 20 00 ...           (22 bytes)
with BOM   FE FF + UTF-16BE, or FF FE + UTF-16LE       (24 bytes)
```

Code pages, and two failures that look alike

Before Unicode, and in many systems still running, text was stored in a **code page**. The single-byte code pages that matter here are tables of at most 256 characters, one byte each; Windows also has double-byte code pages for East Asian scripts. Microsoft's list names code page 1252 "ANSI Latin 1; Western European (Windows)" and 1254 "ANSI Turkish; Turkish (Windows)". Microsoft's own mapping tables, published by the Unicode Consortium, show the problem for the guest's name at once. The 1254 table maps byte FD to U+0131, the dotless ı, and FE to U+015F, ş. The 1252 table has no entry for either. A Turkish name fits in the Turkish code page and does not fit in the Western one, and the hotel's property system exported in 1252.

Two different failures follow from that one fact, and they need different repairs, so they need different names.

Lossy transcoding happens when text is converted into a code page that lacks some of its characters. The converter has three choices for a character the code page lacks. It can stop. It can write the code page's default character, a question mark, byte 3F, named as the default in Microsoft's best-fit file for code page 1252. Or it can use best fit and write a look-alike letter. With the default character, Işıl Yılmaz becomes I??l Y?lmaz: three characters, one ş and two dotless ı, are gone. No later step can tell whether each question mark was ş, ı, ğ or a Greek letter. Microsoft's documentation for its conversion function is blunt: data converted from UTF-16 to non-Unicode encodings "is subject to data loss". Its SQL Server documentation adds that "data loss during code page translations isn't reported".

Windows uses best fit unless told otherwise, and the difference matters. Microsoft's best-fit table for code page 1252 maps ş to a plain s and ı to a plain i, so the default conversion writes Isil Yilmaz: closer to the truth, and a silent change of letters with no question mark to warn anyone. Microsoft advises that "for strings that require validation, such as file, resource, and user names, the application should always use the WC_NO_BEST_FIT_CHARS flag", which turns best fit off. The property system on the night did use the default character, which at least left a visible scar. The Type Contract should name which of the three behaviours a conversion uses: stop, replace, or best fit.

Mojibake is the opposite failure. Nothing is converted; nothing is lost. Correct bytes are decoded with the wrong table. When a later booking-side file, written correctly in UTF-8, was opened by a tool that assumed Windows-1252, each byte was read as one 1252 character. The two bytes of ş, C5 9F, became Å and Ÿ; the two bytes of ı, C4 B1, became Ä and ±. Computed with Python 3.13 and checked against the 1252 table:

```
UTF-8 bytes   49 C5 9F C4 B1 6C 20 59 C4 B1 6C 6D 61 7A
read as 1252  I  Å  Ÿ  Ä  ±  l     Y  Ä  ±  l  m  a  z
result        IÅŸÄ±l YÄ±lmaz        (14 characters, not 11)
and back      encode as 1252, decode as UTF-8: Işıl Yılmaz
```

Mojibake is repairable while the bytes are intact and every byte has a character in the wrong table: encode the garbled text back to 1252 and decode the result as UTF-8. Code page 1252 leaves five bytes undefined (81, 8D, 8F, 90 and 9D), so a UTF-8 character that contains one, such as ŝ (C5 9D), is replaced or rejected on the wrong read and does not come back; the guest's bytes all map. Lossy transcoding is not repairable from the data at all; the original has to be fetched again from the source that still holds it. That is why the two must never share one data-quality label such as "bad characters". The same pattern runs the other way: bytes written correctly in 1254 and read as 1252 come out as Iþýl Ýylmaz, because bytes FD and FE mean ý and þ in the Western table. Figure 5.1 puts the four forms of the night side by side, with the booking spelling shown both in UTF-8 and in UTF-16.

FIGURE 5.1

One name, four forms in five rows: one loses letters, one mislabels the bytes, one transliterates, and no two match byte for byte

The guest's name as typed on the EU booking site: "Işıl Yılmaz", 11 code points

UTF-8 bytes booking site, 14 bytes	`49 C5 9F C4 B1 6C 20 59 C4 B1 6C 6D 61 7A`	ş and ı take two bytes each; the ASCII letters keep one
UTF-16 code units 11 units, 22 bytes	`0049 015F 0131 006C 0020 0059 0131 006C ...`	Big-endian shown; a BOM FE FF or FF FE says which order follows
Windows-1252, '?' property export	`I??l Y?lmaz 49 3F 3F 6C ...`	No ş or ı in the code page: each becomes the default character
Mojibake UTF-8 read as 1252	`IÅŸÄ±l YÄ±lmaz`	No byte lost: 14 bytes decoded with the wrong table, 14 characters
Passport MRZ name field, 39 positions	`YILMAZ<<ISIL<<<<<<<<<<<<<<<<<<<<<<<<<`	ICAO transliteration: Ş to S, ı to I; upper case, no diacritics, < fillers

The replacement row lost three characters (one ş, two ı) for good; the mojibake row lost nothing but the label, so its bytes can be decoded again correctly; the MRZ form is correct by its own rules and still matches neither.

Author's computation, 20 November 2026, Python 3.13.3 codecs; Microsoft code page tables CP1252.TXT and CP1254.TXT and best-fit file bestfit1252.txt (unicode.org); ICAO Doc 9303 Part 3, section 6.A, and Part 4, Eighth Edition 2021. The guest is invented.

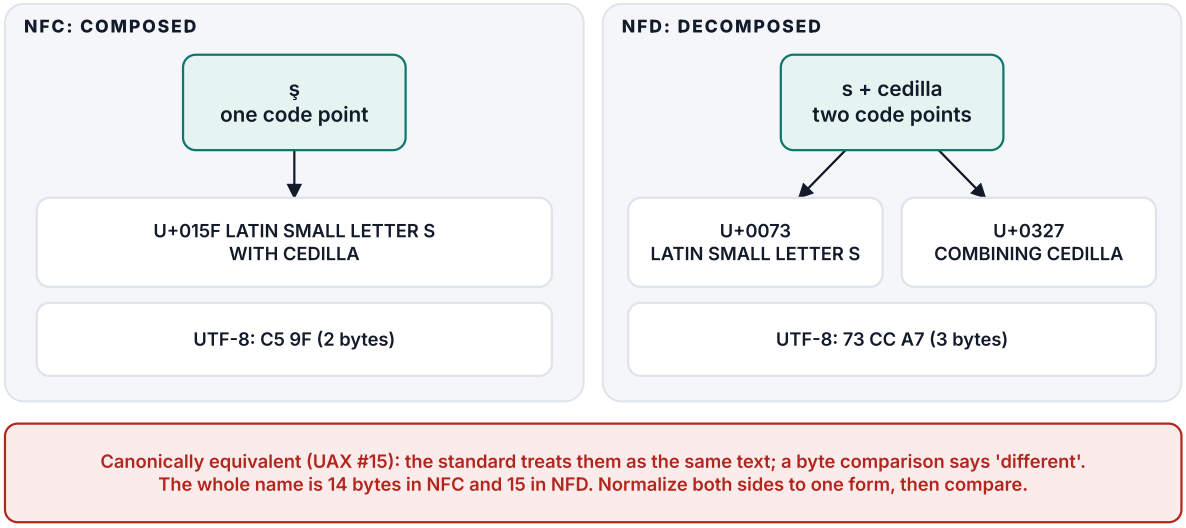
The row that gets the least attention is the last one, because it is correct. A passport's machine readable zone (MRZ) cannot hold ş or ı at all, and it is not supposed to. That rule belongs to identity documents, and it comes later in this chapter.

The same text, different code points: normalization, case and order

Unicode sometimes offers two correct ways to write the same letter. The ş in the guest's first name can be stored as the single code point U+015F or as a plain s, U+0073, followed by U+0327 COMBINING CEDILLA. The two are **canonically equivalent**: the standard treats them as the same text. Their bytes differ, so a byte comparison, a hash, a database unique key or a join treats them as two different names. Figure 5.2 shows the anatomy.

FIGURE 5.2

NFC and NFD: one letter, one or two code points, two or three bytes



Both forms are valid Unicode for the letter in the guest's first name; a system that compares bytes without normalizing treats them as two different names.

Author's diagram based on Unicode Standard Annex #15, Unicode Normalization Forms, revision 58 (Unicode 18.0.0); bytes computed with Python 3.13.3 unicodedata.

Unicode Standard Annex #15, Unicode Normalization Forms (revision 58, for Unicode 18.0.0), defines four forms. **NFD** is canonical decomposition: every letter split into its base and combining marks. **NFC** is canonical decomposition followed by canonical composition: put back together, using precomposed characters wherever they exist. **NFKD** and **NFKC** do the same with compatibility decomposition, which the annex calls "a weaker type of equivalence": under them the "fi" ligature becomes the two letters f and i, and full-width forms become their ordinary counterparts. This book recommends NFC for storage and comparison, because it keeps almost all text as typed (a few singletons change, such as U+212B ANGSTROM SIGN, which becomes U+00C5), and NFKC only for matching and search keys, where a loss of formatting is acceptable; it should not overwrite the stored value, because it changes some text in ways a person would notice.

Normalization also limits the cheap trick of "removing accents" before matching. Decompose the name to NFD, drop the combining marks, and ş becomes s. The dotless ı does not change at all: it has no decomposition, because it is a letter of its own, not an i with something removed. "Işıl Yılmaz" becomes "Isıl Yılmaz", which still matches neither the export nor the passport.

Case mapping is the second trap, and Turkish is the textbook case. In English, i and I are a pair. In Turkish there are two pairs: dotted i with dotted İ, and dotless ı with dotless I. The Unicode Character Database records this in SpecialCasing.txt, whose Turkish and Azeri section begins: "I and i-dotless; I-dot and i are case pairs in Turkish and Azeri". With the default, language-neutral mapping that most libraries use, uppercasing "Işıl" gives "IŞIL" and lowercasing "YILMAZ" gives "yilmaz", with a dotted i the name never had. With the Turkish mapping, lowercasing "YILMAZ" gives "yılmaz" and uppercasing "i" gives "İ". A "case-insensitive" comparison is therefore not one rule but a choice of rules, and it has to be declared. On the night, a case-insensitive match of the booking's spelling against the hospital's compared "işıl" with "isil" and failed, correctly by its own rules.

Collation, the order in which text sorts, is the third. Unicode Technical Standard #10, the Unicode Collation Algorithm (version 18.0.0, 31 August 2026), opens with the problem: "Collation varies according to language and culture: Germans, French and Swedes sort the same characters differently." It can vary by application too: the standard notes that dictionaries and phonebooks may sort differently within one language. The algorithm provides a default table (the Default Unicode Collation Element Table, DUCET), and the Unicode CLDR project tailors it per language. For data, the consequence is practical: a range query, a deduplication pass that compares neighbours after sorting, or a pagination cursor gives different answers under different collations. The collation is part of the contract wherever order is used to make a decision.

Bytes, characters and names

A field length is a number with a unit, and the unit is often wrong. Microsoft's SQL Server documentation calls it "a common misconception" that the n in char(n) and varchar(n) is a number of characters: "n defines the string length in bytes", and "n never defines numbers of characters that can be stored". In a UTF-8 column, the guest's 11-character name needs 14 bytes; in NFD it needs 15. A field declared as nine bytes cuts the name inside the two bytes of the second ı:

```
first 9 bytes  49 C5 9F C4 B1 6C 20 59 C4
decoded        Işıl Y + an incomplete sequence (C4 alone)
strict reader  rejects the value as invalid UTF-8
lenient reader Işıl Y followed by U+FFFD, the replacement mark
```

A truncation that respects bytes but not characters creates invalid text, and the next reader either rejects the record or quietly adds a replacement mark. Either way, two systems now hold different names. The Type Contract states the length unit (code points of the NFC form, or user-perceived characters, the grapheme clusters of Unicode Standard Annex #29), the encoding, and what happens to a value that is too long: reject, or cut at a grapheme boundary and flag the record.

Names are where all of this meets people. The W3C Internationalization article "Personal names around the world", by Richard Ishida, is guidance rather than a standard, and it is the clearest short text on the subject. It advises using a Unicode encoding "in your pages, your back end databases" and elsewhere; not to "assume that the order they will provide names in will be given followed by family"; to "make input fields long enough to enter long names"; and it warns that a four-character Japanese name in UTF-8 is "likely to actually need 12" bytes. The article also suggests asking for a name in both native script and Latin-only form when a Latin form is needed, rather than deriving one silently.

Identity documents make that derivation explicit. ICAO Doc 9303, Machine Readable Travel Documents, Part 3 (Eighth Edition, 2021), section 4.6, requires names in the MRZ in "upper-case OCR-B characters ... without diacritical marks": the issuing State transliterates national characters, writes the primary identifier, two filler characters "<<", then the secondary identifier, and fills the rest of the line with "<". Its section 6.A lists the recommended transliterations: Ş to S, İ to I, and U+0131, listed as "I without dot (Turkey)", to I. Part 4 gives the name 39 positions, 6 to 44, on the first line of a passport's MRZ. So the guest's passport reads:

```
name field   YILMAZ<<ISIL<<<<<<<<<<<<<<<<<<<<<<<<<
             primary  <<  secondary, then fillers to 39
```

The MRZ form is correct, standard and lossy by design. It is a key for machines at a border, not the person's name. The same table offers choices for some letters (Ö may become "OE or O", Ü "UE or UXX or U"), so two documents can transliterate one name differently. A system that registers patients or guests from a passport should store the visual zone's name as a labelled document name, and the MRZ form as a separate, labelled match key.

Text as attack surface

Everything that makes two different strings look the same also lets an attacker make a hostile string look harmless. Unicode Technical Standard #39, Unicode Security Mechanisms (version 18.0.0, 27 August 2026), defines the tools. Two strings are **confusable** when their skeletons are equal: the skeleton is computed by decomposing to NFD, removing default-ignorable code points and mapping each character to a prototype from the standard's confusables.txt data. The standard's own example of mixed-script confusables is "paypal" and "paypal", where the second word uses U+0430 CYRILLIC SMALL LETTER A. A payee name, a vendor master record, a user name or a domain in an approval email can carry the same trick.

Three controls follow from this chapter. First, use skeletons and the mixed-script test that UTS #39 defines to flag possible impersonation for review, not to decide identity: equal skeletons can belong to two different names, and identity comes from authoritative keys and

labelled name variants. Second, never let best-fit conversion touch validated strings: Microsoft's own example is "∞" (infinity) mapping to the digit 8 in some code pages, and a check made before conversion no longer holds after it. Third, treat a failed decode as an event, not as a warning in a log; a file that cannot be decoded as its declared encoding was either mislabelled or tampered with. The payment side of the night, where the guest's name meets a bank's character-set profile, is told in Chapter Thirteen.

TYPE CONTRACT: TEXT

1. Type, precision and rounding. A character string; maximum length stated in a named unit (code points of the NFC form, or grapheme clusters), with the byte limit of each store beside it; the rule for an over-long value (reject, or cut at a grapheme boundary and flag). *Test:* a value longer than the limit, or one that ends inside a multi-byte sequence, is held.

2. Encoding and serialization. UTF-8 at every boundary; normalization form NFC; whether the producer writes a BOM (readers accept and discard one); for UTF-16, the byte order. Any conversion to a code page names its behaviour: stop, replace, or best fit, and replace or best fit is allowed only for display copies. *Test:* the bytes decode strictly as UTF-8 and equal their own NFC form; a conversion that used a default or best-fit character fails the load.

3. Unit or currency. Not applicable. *Test:* none beyond line 1's length unit.

4. Time basis. Not applicable to the characters; for names, the value as captured at the source, with its capture date. *Test:* a changed name arrives as a new version, not an overwrite.

5. Identity scope. Which name of which person: the name as the person writes it, a Latin form if requested, and the MRZ form as a separate match key; the language whose case and collation rules apply (Turkish for this guest). *Test:* matching on a name uses the declared comparison (NFC or NFKC, case folding with the named language) and never raw bytes; skeletons only raise flags for review.

6. Absence. Empty string, null and a single filler character mean different things: not given, not captured, deliberately absent (as an MRZ filler). *Test:* a value made only of question marks, U+FFFD or fillers is held as a failed conversion, not accepted as a name.

7. Authority. The owner of each source (the guest data team for the booking, hospital registration for the patient record); the Unicode version of the validating library; for MRZ data, ICAO Doc 9303 Part 3, Eighth Edition. *Test:* the Unicode

version and the transliteration table travel as metadata, and a library older than the data's version is reported.

On the night, the guest's name travelled three routes at once. The EU booking site stored it in UTF-8, correctly. The hotel's property system exported guest records in Windows-1252 for the claim file, and its converter replaced the two characters it could not map, so the claim file received I??l Y?lmaz. At the hospital, the registration desk scanned the passport and took the name from its machine readable zone, YILMAZ<<ISIL, so the patient record carried the transliterated YILMAZ and ISIL. A later UTF-8 file from the booking side was opened as Windows-1252 and showed a fourth form, IÅŸÄ±l YÄ±lmaz. Exact matching failed, case-insensitive matching failed, and the adjuster spent about an hour confirming by hand that the three records were one person (an illustrative figure). The loss code is IDENTITY: no system dropped the guest, but no key could join her records.

THE HOP · 3 · 2025 · EU BOOKING SITE (UTF-8) TO PROPERTY SYSTEM EXPORT (WINDOWS-1252 CSV), THEN THE CLAIM FILE; HOSPITAL REGISTRATION READS THE PASSPORT'S MACHINE READABLE ZONE

Record	Guest name (One Night, Every Value, illustrative), hop 3, 2025
From	EU booking site (UTF-8)
To	Property system export (Windows-1252 CSV), then the claim file; hospital registration reads the passport's machine readable zone
Mechanism and schedule	The guest's name. The property system's legacy export converts text to Windows-1252; the hospital's registration desk scans the passport and takes the name from its machine readable zone
Format change; what was lost or assumed	IDENTITY The guest's UTF-8 name (initials I. Y.) loses its three non-ASCII letters (one ş, two ı) in Windows-1252, each replaced by '?'; the passport MRZ gives the name in upper case, surname first, transliterated. **Three spellings of one person; exact and case-insensitive matching both fail; a later UTF-8 file opened as Windows-1252 shows a fourth, mojibake form**
Accountable owner	Group guest data team (booking); hospital registration (patient)
Who can change it	Front desk and booking staff; hospital registration clerks
Classification; retention or erasure duty	Personal data (GDPR for the EU booking); protected health information at the hospital
Read by (people, apps, models, agents)	Front desk, finance, the claim file, the payment bank
Cost per month; basis	One-off: a manual identity check by the adjuster, about an hour (not monthly) (illustrative)

OUR VIEW, AS OF NOVEMBER 2026

The encoding war is largely over at the boundary: UTF-8 has won for files, messages and APIs. The failures that remain sit where it meets older code pages, where comparisons ignore normalization and language, and where field lengths count bytes. A language model reading the claim file will often read I??I Y?Imaz and IÅŸÄ±I YÄ±Imaz as the same person, and sometimes it will be right. A contract that labels each form, and says which failure produced it, lets the model say why instead of guessing.

ERA CARD

Reader. From the card reader with a six-bit code to the model with a tokenizer, every reader of text has needed one rule the bytes do not carry: which table turns them into letters.

Security. Confusable characters, best-fit conversion and lenient decoding let a hostile string pass a check made on a different form; flag confusables by skeleton, disable best fit for validated fields, and treat decode failures as events.

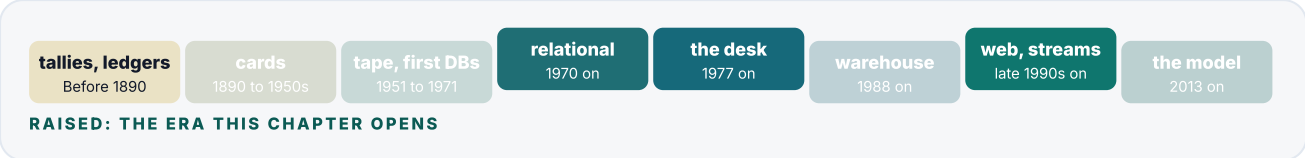
Cost. A lossy conversion is paid for later, by a person who fetches the original again; on the night, an adjuster's hour per claim (illustrative), multiplied by every name a legacy export touches.

AI. Models are tolerant readers of damaged text, which hides failures as often as it repairs them; give the model the encoding, the normalization form and the provenance of each name form, and ask it to say when two forms only might be one person.

Monday rule: find every export that still writes a single-byte code page, set it to stop rather than replace or best-fit when a character does not map, and make the next load hold any name whose conversion reported a default character, or that contains U+FFFD, and compare it with the source record.

CHAPTER SIX

Time



At 02:00 on Sunday 2 November 2025, clocks in America/New_York went back to 01:00, and every reading from 01:00:00 to 01:59:59 happened twice. In the first pass of that hour, at 01:42:10 Eastern Daylight Time, fire panel 2 in the hotel's kitchen wing went into alarm. In the second pass, at 01:20 Eastern Standard Time, the guest arrived at the health system's emergency department. Eight days later the insurer's claims-triage agent read the two times as text and wrote that treatment was recorded 22 minutes before the fire alarm. In UTC the arrival came 37 minutes and 50 seconds after the alarm. No system altered a clock digit. Two kinds of time were treated as one. The decision this chapter changes: **every time that crosses a boundary says which kind of time it is, and every event timestamp carries its offset and is compared only as an instant**.

Five kinds of time

Most systems have one column type called "timestamp" and several meanings hiding inside it. The Type Contract's fourth line, the time basis, starts by naming which kind of time a value is. Five kinds cover nearly every field of the night.

Kind	What it is	From the night	Safe to compare with another of its kind?
Instant	One point on the global time line, written in UTC or with its offset from UTC	2025-11-02T05:42:13Z, the station's receipt	Yes, always
Local wall time	What a clock on a wall showed, with no offset	2025-11-02 01:42:10, the panel log	No, not across zones or a clock change
Date	A calendar day, with no time and no offset	2 November 2025, the date of loss on the claim	Yes, as days; never as instants
Duration	An amount of elapsed time, or a calendar amount such as "one month"	37 minutes 50 seconds, alarm to arrival	Elapsed, yes; calendar amounts depend on where they start
Zoned time	An instant plus the name of the zone whose rules give its local meaning	2025-11-02T01:42:10-04:00[America/New_York]	Yes as instants; the zone adds local meaning

Two words do most of the damage, so they need definitions. An **offset** is a number: the difference between local time and UTC at one moment, such as minus 4 hours for 01:42:10 EDT on that night. A **zone** is a set of rules with a name, such as America/New_York, that says which offset applies at each moment since 1970 (and, for many places, earlier) and, as far as anyone knows, in the future. An offset fixes an instant. A zone name says how people in a place read that instant. "EST" is an abbreviation, not a zone name: the same letters are used in more than one place, and software differs on whether they mean a fixed UTC minus 5 or "Eastern time, whichever applies". On the night it was still enough to place 01:20 in the second pass, and the extraction dropped it. The night needed both kinds of fact. The offset would have told the triage agent which pass of the repeated hour each time belonged to; the zone name tells a later reader that the place was New York and that its rules changed that night.

The kinds do not convert into each other for free. An instant becomes a local wall time only through a zone's rules. A local wall time becomes an instant only if the reader knows the zone and, inside a repeated hour, which pass was meant. Some wall times have no instant at all: on the spring-forward night the readings from 02:00 to 02:59 never happen in New York. A date is not midnight. *From Tally to Token* told how a time zone mistake moved

laboratory results and how a century window misread two-digit years, and it separated event time from processing time; this chapter goes inside the single time value those stories assumed.

The repeated hour: hops 1 and 2

The main line of the night starts with two time fields, born in two buildings, that met in one claim file.

Hop 1, the alarm time. Fire panel 2 logs events in local wall time and writes no offset. The facilities vendor's nightly export sends the panel's events as text, and the claim file imports that export when the adjuster requests it. The field arrives as "2025-11-02 01:42:10". On almost any other night that text would name one instant. On this night it names two, an hour apart: 05:42:10 UTC if it was the first pass, under EDT, or 06:42:10 UTC if it was the second, under EST. The export says neither the pass nor the zone. The offset was never written down, so it was never lost in transit; it was missing from birth.

THE HOP · 1 · 2025 · FIRE PANEL 2 EVENT LOG TO FACILITIES SYSTEM, THEN THE CLAIM FILE (CSV EXPORT)

Record	Alarm time (One Night, Every Value, illustrative), hop 1, 2025
From	Fire panel 2 event log
To	Facilities system, then the claim file (CSV export)
Mechanism and schedule	The alarm time. The facilities vendor's nightly export sends panel events as text; the claim file imports the export on request
Format change; what was lost or assumed	TIME Panel wall time '2025-11-02 01:42:10' as text; no offset, no zone name. **Inside the repeated hour the text names two instants an hour apart; the export says neither which pass nor which zone**
Accountable owner	Hotel facilities manager
Who can change it	Facilities vendor technicians (panel clock and configuration); the export job's service account
Classification; retention or erasure duty	Operational safety record; kept for the claim and by fire code
Read by (people, apps, models, agents)	Facilities staff, the fire investigator, the insurer's adjuster and its triage agent
Cost per month; basis	Monthly: negligible to store. One-off: two hours of a handler's time on the wrong flag (hop 14) (not monthly) (illustrative)

Hop 2, the arrival time. The hospital kept the offset. Its registration system recorded the arrival in HL7 version 2 with the offset, and its FHIR services kept it too. The guest's records left the hospital as PDF documents, released to the insurer under her written authorisation for a liability claim; no feed or interface connects the two companies. The PDF prints "11/02/2025 01:20 EST". The insurer's document-extraction service reads the page and fills the claim field with "2025-11-02 01:20". It keeps the clock reading and drops the three letters that said which pass of the hour this was.

THE HOP · 2 · 2025 · HOSPITAL REGISTRATION (HL7 V2, OFFSET KEPT; FHIR INSIDE THE HOSPITAL, OFFSET KEPT) TO PDF RECORDS RELEASED TO THE INSURER, THEN ITS DOCUMENT-EXTRACTION SERVICE

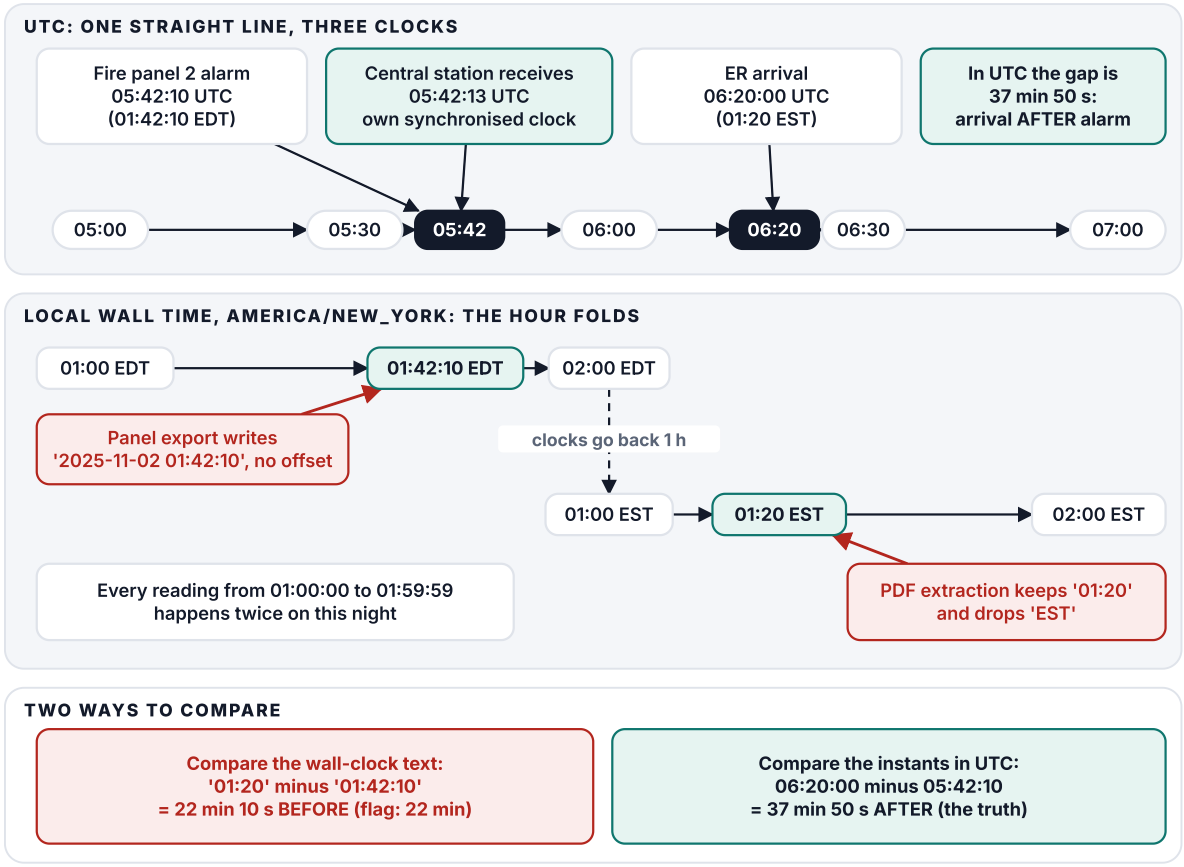
Record	Emergency department arrival time (One Night, Every Value, illustrative), hop 2, 2025
From	Hospital registration (HL7 v2, offset kept; FHIR inside the hospital, offset kept)
To	PDF records released to the insurer, then its document-extraction service
Mechanism and schedule	The arrival time. Released under the guest's written authorisation; the extraction service reads the PDF and fills claim fields
Format change; what was lost or assumed	TIME RIGHTS Printed '11/02/2025 01:20 EST' becomes the field value '2025-11-02 01:20'. **The extraction keeps the clock reading and drops 'EST'; the authorisation's description and purpose do not travel with the extracted fields, so later readers cannot tell what the release allowed**
Accountable owner	Hospital health information management (source); insurer claims operations (extracted fields)
Who can change it	Registration clerks (source); the extraction service and claims staff (claim fields)
Classification; retention or erasure duty	Protected health information at the hospital, where 45 CFR 164.508 governs the disclosure; at the insurer, state insurance privacy law (many states follow NAIC model regulation 672) governs its use
Read by (people, apps, models, agents)	Clinicians; then the adjuster, the nurse reviewer and the triage agent
Cost per month; basis	Monthly: per-page extraction fees plus human review for each claim file (illustrative)

Hop 2 carries a second loss, of rights rather than time. Under HIPAA, 45 CFR 164.508 governs what the hospital may disclose: a valid authorisation must describe "the information to be used or disclosed" in "a specific and meaningful fashion", name the recipient and state the purpose, and it must warn that information disclosed under it may be redisclosed by the recipient "and no longer be protected by this subpart" (section 164.508(c)). That rule binds the hospital as the disclosing covered entity; whether the release itself went beyond the authorisation is the hospital's question. The insurer is not a covered entity in this transaction. What governs it is insurance privacy law: many states follow the NAIC's model regulation 672, written under the Gramm-Leach-Bliley Act, which applies to "all nonpublic personal health information" a licensee holds and forbids disclosing it without the person's authorisation, with exceptions for functions such as claims administration and fraud detection. The extraction service loaded every page into general claim fields, and every claim reader, the triage agent included, could see them. The authorisation's description and purpose did not travel as a field, so nothing downstream could check a use against them.

Figure 6.1 puts the three clocks of the night on one line. On the UTC axis the alarm, the station's receipt and the arrival fall in order: 05:42:10, 05:42:13 and 06:20:00. On the local axis the hour folds: the alarm sits in the first pass, before the clocks go back, and the arrival sits in the second.

FIGURE 6.1

The repeated hour: in UTC the three clocks sit on one straight line; in local wall time the hour folds and 01:20 looks earlier than 01:42



On 2 November 2025 the panel alarm and the emergency department arrival are 37 minutes 50 seconds apart in UTC, with the station's receipt three seconds after the alarm, but compared as wall-clock text the arrival appears 22 minutes 10 seconds before the alarm.

Author's diagram based on the illustrative thread of this book; offsets computed with Python zoneinfo and the IANA time zone database (America/New_York).

The triage agent compared the two strings. "01:20" minus "01:42:10" is minus 22 minutes and 10 seconds, so it wrote "22 minutes before". It was not careless arithmetic. It was correct arithmetic on the wrong kind of value. The central monitoring station settled the question: it received the alarm signal at 05:42:13 UTC on its own synchronised clock, three seconds after the panel's alarm when that is read as EDT, and nearly an hour before the panel's reading would fall if read as EST. A third clock, with a known basis, is what let a claims handler correct the flag on 11 November. This book measures the gap from the panel's reading: 37 minutes 50 seconds, rounded to 38 minutes in plain sentences (37 minutes 47 seconds from the station's receipt); the agent's 22 minutes is the rounded form of 22 minutes 10 seconds.

The same trap exists in ordinary code. In Python's datetime, the fold attribute marks which pass of a repeated hour a wall time belongs to; the documentation says the values 0 and 1 "represent, respectively, the earlier and later of the two moments with the same wall time representation". But when both values carry the same zone object, Python's subtraction ignores the zone: "No time zone adjustments are done in this case", and comparison ignores fold. The check below, run for this book in Python 3.13 with the IANA data, shows both results.

```
from datetime import datetime, timezone
from zoneinfo import ZoneInfo
ny = ZoneInfo("America/New_York")
alarm = datetime(2025, 11, 2, 1, 42, 10, tzinfo=ny, fold=0)
er = datetime(2025, 11, 2, 1, 20, tzinfo=ny, fold=1)
alarm.astimezone(timezone.utc)  # 05:42:10+00:00, EDT
er.astimezone(timezone.utc)     # 06:20:00+00:00, EST
er - alarm                      # -1 day, 23:37:50 (-22:10)
er.astimezone(timezone.utc) - alarm.astimezone(timezone.utc)
                                # 0:37:50
```

Even with the pass recorded correctly, a comparison in local time gives the agent's answer. Converted to UTC first, it gives the truth. The rule for code, warehouses and agents is the same: convert to instants, then compare.

Who decides what the clock says

The offset of a place is set by law, and law changes. In the United States the federal rule is in 15 U.S.C. 260a: daylight saving time runs from "2 o'clock antemeridian on the second Sunday of March" to "2 o'clock antemeridian on the first Sunday of November", and a state may exempt itself by law: a state entirely within one time zone as a whole, and a state in more than one zone either as a whole or for the entire area lying within one zone. Those dates come from section 110 of the Energy Policy Act of 2005 (Public Law 109-58), which substituted "second Sunday of March" for "first Sunday of April" and "first Sunday of November" for "last Sunday of October". The amendment took effect one year after enactment on 8 August 2005 or on 1 March 2007, whichever was later, so the current dates have applied since 2007. A system that stored local times before 2007 and converts them with today's recurring rule, instead of the dated history the tz database keeps, gets some of them wrong by an hour.

As of November 2026 the rule has not changed. On 14 July 2026 the House of Representatives passed H.R. 139, the Sunshine Protection Act of 2025, by 308 votes to 117 (Roll Call 238). The bill would repeal section 260a's temporary period and make daylight saving time permanent. It was received in the Senate on 15 July 2026 and referred to the

Committee on Commerce, Science, and Transportation, and Congress.gov records no later action as of 20 November 2026. It is not law. If it becomes law, local times in New York will map to different instants from its effective date, and every stored wall time from before it will still need the old rules.

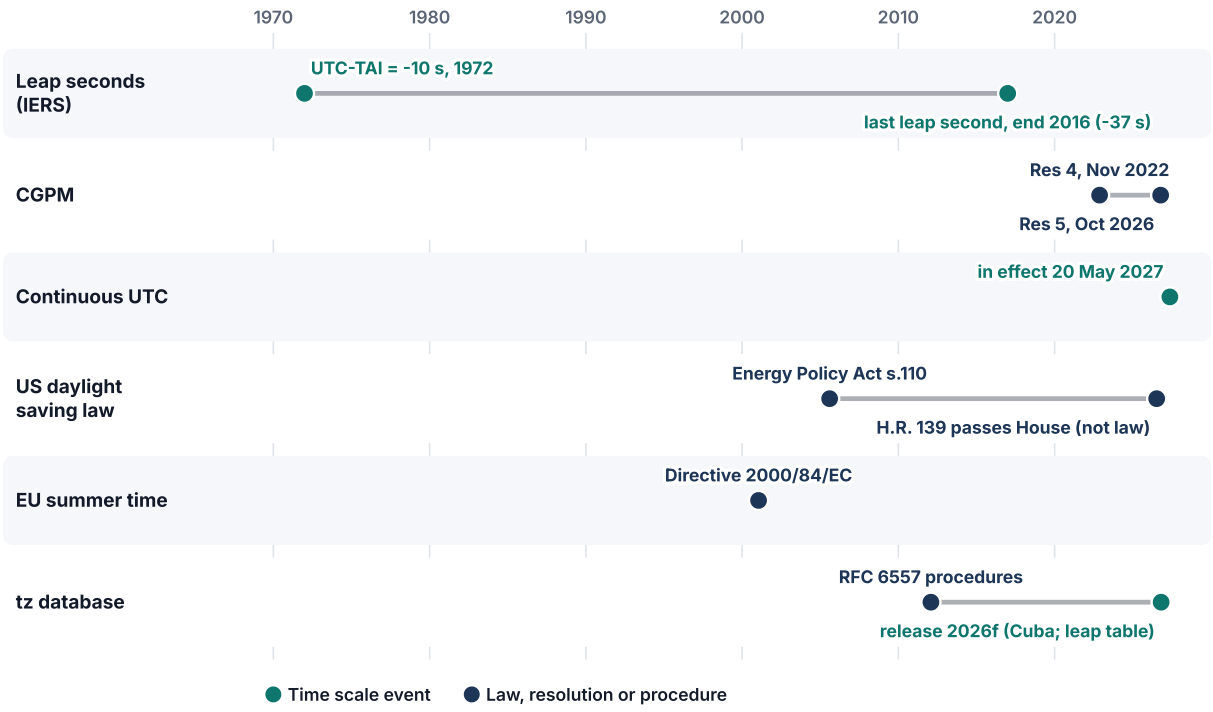
Europe changes its clocks on a different day. Directive 2000/84/EC of 19 January 2001 says that the summer-time period ends "in every Member State, at 1.00 a.m., Greenwich Mean Time, on the last Sunday in October". In 2025 that was 26 October. For one week, from 26 October to 2 November 2025, Paris was five hours ahead of New York instead of six. The guest booked on the group's EU site; any system that converted between the two with a fixed six-hour difference was wrong that week by an hour. Offsets are not properties of a pair of cities. They are properties of instants.

Software learns these rules from one shared source. The IANA Time Zone Database, often called tz or zoneinfo, "contains code and data that represent the history of local time for many representative locations worldwide", and IANA says it is "updated periodically to reflect changes made by political bodies to time zone boundaries, UTC offsets, and daylight-saving rules". RFC 6557 (February 2012, BCP 175), "Procedures for Maintaining the Time Zone Database", sets out how updates are submitted, how decisions to include them are made, and how a designated expert is selected for the time zone community. The notice can be short. Release 2026f, published on 27 October 2026, recorded that Cuba "will not fall back to -05 on 2026-11-01 and instead plans to maintain -04 through 2027-11-07": five days between the release and the change. A server that had not updated its tz data by 1 November converted Cuban local times with the wrong offset.

That is why a zone name and an offset do different jobs. RFC 9557 (April 2024) says that a named IANA zone in a timestamp "implies that the intent is for the rules that are current at the time of interpretation to apply". The name follows the law; the offset records what was true when the value was written. A contract that keeps both can detect when they disagree, and RFC 9557 calls such a timestamp inconsistent.

FIGURE 6.2

The rules behind a timestamp keep changing: leap seconds end by decision of the CGPM, while zone and daylight saving rules change by law, sometimes at a few days' notice



Twenty-seven leap seconds were inserted from 1972 to the end of 2016 and none since, and continuous UTC takes effect on 20 May 2027, while US, EU and other zone rules changed on their own schedules and reach software through the tz database.

Source: IERS Leap_Second.dat and Bulletin C 72 (July 2026); BIPM, CGPM 2022 Resolution 4 and CGPM 2026 Resolution 5; 15 U.S.C. 260a and notes; House Clerk Roll Call 238 (2026); Directive 2000/84/EC; RFC 6557; IANA tz NEWS, release 2026f.

UTC, TAI and the end of the leap second

Instants need a reference scale. International Atomic Time (TAI) counts SI seconds from atomic clocks and never adjusts. Coordinated Universal Time (UTC) is the scale civil time is built on. In the words the 28th General Conference on Weights and Measures (CGPM) recalled from its 2018 resolution, UTC is "a time scale produced by the International Bureau of Weights and Measures (BIPM) with the same rate as International Atomic Time (TAI), but differing from TAI only by an integral number of seconds", and it is "the only recommended time scale for international reference and the basis of civil time in most countries". The integral number exists to keep UTC near UT1, the time scale that follows the rotation of the Earth. When the gap grew, a leap second was inserted. The International Earth Rotation and Reference Systems Service (IERS) records UTC minus TAI as minus 10

seconds on 1 January 1972 and minus 37 seconds from 1 January 2017: 27 leap seconds in all, the last one at the end of 2016. Its Bulletin C 72 of 6 July 2026 states: "NO leap second will be introduced at the end of December 2026."

The CGPM has taken two decisions, four years apart, and they say different things. **Resolution 4 of the 27th CGPM (2022)**, "On the use and future development of UTC", "decides that the maximum value for the difference (UT1-UTC) will be increased in, or before, 2035". It asked the International Committee for Weights and Measures to propose a new maximum and to "draft a resolution including these proposals for agreement at the 28th meeting of the CGPM (2026)". It did not abolish the leap second and did not set a date. **Resolution 5 of the 28th CGPM (2026)**, "On the technical actions needed to ensure the continuity of UTC", adopted at the meeting of 13 to 15 October 2026, "decides that continuous UTC will become effective on 20 May 2027" and that "the maximum value for the difference |UT1-UTC| will be 3 600 seconds (1 hour), ensuring the long term continuity for UTC for several centuries". Among its reasons it notes that a negative leap second "has not previously been applied, and it is considered to pose a high risk of causing anomalies and disruption to critical infrastructures". The tz database's release 2026f followed: "No more leap seconds, and leap second tables no longer expire."

For a data architect the practical points are three. First, UTC minus TAI stays at minus 37 seconds, with no change scheduled, so a system that stores instants in UTC does not need a leap second table for future values. Second, history keeps its 27 leap seconds. POSIX time does not count them: the standard says that in seconds since the Epoch "each and every day shall be accounted for by exactly 86400 seconds", so subtracting two Unix times across a leap second gives an answer one second short of elapsed SI time. RFC 3339 allows a seconds field of 60 because leap seconds exist. Third, a contract that needs elapsed time to the second over long spans should say which scale it uses.

Text forms, epochs and the clocks behind them

A time value crosses a boundary as text or as a number from an epoch. The text forms come in three layers. ISO 8601-1:2019, "Date and time: Representations for information interchange, Part 1: Basic rules", is the international standard. RFC 3339 (July 2002) is, in its own words, "a profile of the ISO 8601 standard" for timestamps in Internet protocols: a full date, the letter T, a time and a required offset, as in 2025-11-02T01:42:10-04:00. Its section 4.4 explains why the offset is required: "interpretation of an unqualified local time zone will fail in approximately 23/24 of the globe", and systems that keep local time and depend on synchronisation "MUST use a mechanism that ensures correct synchronization with UTC", starting with the Network Time Protocol. RFC 9557, "Date and Time on the Internet: Timestamps with Additional Information", adds a suffix in brackets for the zone name, as in 2025-11-02T01:42:10-04:00[America/New_York]. It also updates RFC 3339 on

one point: the offset Z now means "the time in UTC is known, but the offset to local time is unknown", while +00:00 still says that UTC is the preferred reference. A Z on a value tells a reader that the local meaning is missing.

Numbers are shorter and more dangerous, because the epoch and the unit live only in documentation. POSIX defines the Epoch as "zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal Time (UTC)"; the alarm is 1762062130 seconds after it. The Network Time Protocol counts from "0 h 1 January 1900 UTC", its prime epoch, and RFC 5905 notes that its 32-bit seconds field wraps "some time in 2036". Excel's 1900 date system gives January 1, 1900 the serial value 1, and Microsoft documents that "Excel incorrectly assumes that the year 1900 is a leap year", a choice kept for compatibility with Lotus 1-2-3. The serial count therefore includes a 29 February 1900 that never existed; Microsoft notes that correcting it would decrease "almost all dates" in existing worksheets by one day, and that as things stand "the WEEKDAY function returns incorrect values for dates before March 1, 1900". The alarm date is serial 45963. A program that converts serials with its own correct calendar must copy that quirk or be a day out. A number such as 45963 or 1762062130 is a time only for a reader who knows the epoch, the unit and whether it counts leap days and leap seconds the same way.

Widths matter too. A signed 32-bit count of seconds from the Unix Epoch reaches its largest value, 2,147,483,647, at 03:14:07 UTC on 19 January 2038, and the next second wraps to 1901. The Open Group Base Specifications Issue 8 (IEEE Std 1003.1-2024) closed the gap in the standard: "time_t shall be an integer type with a width ... of at least 64 bits". Embedded devices, file formats and database columns that store 32-bit seconds are not covered by that sentence. A fire panel in service for twenty years may be such a device, and its export format is the field's contract whether anyone wrote it down or not.

Every timestamp is also the reading of a particular clock. The Network Time Protocol, version 4, is specified in RFC 5905 (June 2010). It states that with NTPv4 "servers and clients are precise within a few tens of milliseconds with poll intervals up to 36 hours", and that a timestamp of zero is "a special case representing unknown or unsynchronized time". A clock that has never synchronised writes times that look, in the log, exactly as valid as the station's. That is why the station record could settle the night: its clock had a known source and a known basis, and the panel's did not. The time-basis line of a contract names the clock and its synchronisation state as well as the zone.

TYPE CONTRACT: TIME

1. Type, precision and rounding. Name the kind: instant, local wall time, date, duration or zoned time. For an event, instant to the second; by this company's rule, sub-second digits are truncated, so an event is never stamped later than it

happened. *Test:* a value that parses as a different kind (a wall time where an instant is declared) is held.

2. Encoding and serialization. RFC 3339 text in UTF-8 with a numeric offset, plus the RFC 9557 zone suffix where local meaning matters; in a typed store, a timestamp type whose UTC adjustment is declared. An epoch number names its epoch and unit. *Test:* the text matches the RFC 3339 grammar and the offset is present.

3. Unit or currency. Not applicable for an instant. For a duration: the unit, and whether it is elapsed time or a calendar amount. *Test:* a duration without a unit is held.

4. Time basis. The heart of this chapter. Which kind of time; the offset at the moment of the event; the IANA zone name for its local meaning; the clock that produced it and whether that clock was synchronised; event time or recording time. Comparisons and arithmetic are done on instants, in UTC. *Test:* the offset agrees with the zone's rules at that instant; a local time inside a repeated or skipped hour without an offset is held, not guessed; the suite includes 2 November 2025, 01:42:10 against 01:20, and passes only when it answers "ambiguous, need offset" for the text and "37 minutes 50 seconds after" for the instants.

5. Identity scope. Which event of which source: the alarm of fire panel 2 at the hotel, not the time the export ran. *Test:* every time field names its source system and event.

6. Absence. Missing means not captured. Never written as a sentinel such as midnight, 1900-01-01, 1970-01-01 or zero. *Test:* the interface lists the field's forbidden sentinels and holds them; a field whose real values can include 1970-01-01, such as a birth date, does not list it, and an NTP timestamp of zero is read as "unknown or unsynchronized", as RFC 5905 defines it.

7. Authority. The owner of the clock and of the field; the tz release used to convert (for example 2026f); which source wins on conflict (for the night, the central station record). *Test:* the tz release is recorded with each conversion, and the conflict rule is written before the conflict.

OUR VIEW, AS OF NOVEMBER 2026

The end of the leap second removes one hazard from future instants and none from wall times. The risk that grows is political: permanent daylight saving time in the United States, regional changes such as Cuba's, and the long tail of devices that log local time. A company that stores offsets and zone names can absorb those changes. One that stores bare wall times will have to guess what they meant, and so will its agents.

ERA CARD

Reader. Panels, registration systems, document extraction, warehouses and a triage agent; each read "01:42" and "01:20" with a different idea of what kind of time they were.

Security. A clock is an input. An unsynchronised or tampered clock writes valid-looking times, and logs, certificates and audit trails trust them; a contract that names the clock source makes that trust visible.

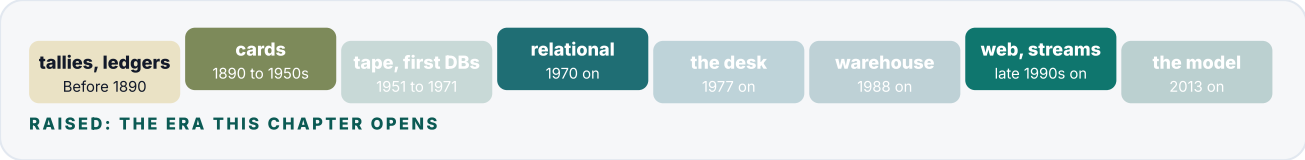
Cost. An offset costs a few bytes per value. Its absence cost two hours of a handler's review, a wrong flag on a person's claim, and a correction that needed a third clock.

AI. A model compares the text it is given. Given two wall times from a fall-back night, it should answer "ambiguous, need offset"; given two instants with their contracts, it gives the right number.

Monday rule: find every boundary field that stores a time without an offset, run the 2 November 2025 test through the reports and agents that compare times, and require RFC 3339 with offset and zone name on every new interface.

CHAPTER SEVEN

Identity: Keys and Identifiers



Before the insurer's triage agent ever compared two times, its claim intake had a simpler question to answer: which policy, and which person? The broker's list gave the policy as 417730, and the policy system had no such number. The date of birth that intake stored for the claimant did not match the hospital's record. Neither value was corrupt. Each had been read by a system that guessed what kind of value it was holding: a number where there was a code, a month where there was a day. An identifier is the value every other value hangs from; when it breaks, the record does not look wrong, it simply cannot be found. The decision this chapter changes: **every identifier that crosses a boundary travels as text with a declared pattern, length and issuer, is checked on arrival, and is matched exactly, never approximately**.

Natural keys and surrogate keys

A **key** is a value that picks out one entity within a scope: one policy among the insurer's policies, one patient among the hospital's patients. A **natural key** is a key that already exists in the world and carries meaning: a passport number, a policy number whose prefix names a series, a person's name and date of birth taken together. A **surrogate key** is a key the system assigns and that means nothing beyond "this row": a sequence number, a random identifier.

Natural keys are convenient because people can read them, and fragile for the same reason. The meaning inside them belongs to someone else and changes on someone else's schedule. A name is spelled three ways in one night (Chapter Five). A date of birth is written in two orders (below). A policy number's prefix records a decision about numbering series that the insurer may revise. Surrogate keys are stable because they mean nothing, but they solve the problem only inside the system that issued them. The moment the record leaves, someone has to map the surrogate back to a natural key, and that map becomes the new boundary.

Two standards make the same point. When the US Centers for Medicare & Medicaid Services set the requirements for the National Provider Identifier (NPI) in January 2004, it wrote that the NPI "will consist of 9 numeric digits followed by one numeric check digit" and "will not have embedded intelligence". RFC 9562, the 2024 standard for universally unique identifiers, advises database designers "to avoid name-based UUID natural keys and, instead, to utilize time-based UUID surrogate keys" (section 6.13). A key with no business meaning inside it cannot be made wrong when that meaning changes.

The enterprise answer is usually both: a surrogate inside each system, a natural key at each boundary, and a written statement of which natural key the boundary uses. The insurer's commercial lines key is the natural key CP 0041-7730. Inside the policy system it has a surrogate. At the broker, at intake and in the claim file, the natural key is the only thing the three parties share, so its form is the contract.

Identifiers are strings, even when they are digits

An identifier made of digits is not a number. Nobody adds two policy numbers, averages them or rounds them. Its digits are characters in fixed positions, and its leading zeros are part of it. A number type throws exactly that information away: 00417730 and 417730 are the same number and two different keys. Long numeric identifiers have a second problem, covered in Chapter Three: a reader that turns a 64-bit id into a binary64 double can change its last digits without any error.

Separators and prefixes are characters too. CP 0041-7730 has a series prefix, a space, eight digits and a hyphen. Which of these are part of the key and which are decoration for human readers is a decision the issuer must write down. If the canonical form is CP00417730, every copy can strip the space and hyphen safely and every matcher can compare canonical forms. If nobody wrote it down, a well-meant clean-up decides for everyone.

Hop 5, the policy number. Before sending the claim, the broker's account team cleaned its list of policy numbers with find-and-replace. The prefix "CP", the space and the hyphen went. What remained, 00417730, looked like a number, and the cell became the number 417730. Claim intake searched for policy 417730 and found nothing. The sheet had no accountable owner: anyone at the broker with the file could change it, and it stayed in use for other claims. The insurer's policy administration owned the source; nobody owned the copy. The claim opened two days late (illustrative).

THE HOP · 5 · 2025 · INSURER'S POLICY SYSTEM TO BROKER'S SPREADSHEET, THEN CLAIM INTAKE

Record	Policy number (One Night, Every Value, illustrative), hop 5, 2025
From	Insurer's policy system
To	Broker's spreadsheet, then claim intake
Mechanism and schedule	The policy number. The broker cleans a list of policy numbers with find-and-replace before sending it
Format change; what was lost or assumed	IDENTITY OWNERSHIP 'CP 0041-7730' loses the prefix, the space and the hyphen; the cell turns '00417730' into the number 417730. **Intake finds no policy 417730; the cleaned sheet has no accountable owner and stays in use for other claims**
Accountable owner	None for the copy (the broker's account team made it); the insurer's policy administration owns the source
Who can change it	Anyone at the broker with the file
Classification; retention or erasure duty	Commercial record
Read by (people, apps, models, agents)	Claim intake, the broker
Cost per month; basis	One-off: a two-day delay in opening the claim (not monthly) (illustrative)

Two losses are recorded, and both matter. IDENTITY, because the key was renumbered by a reader that took it for a quantity. OWNERSHIP, because a copy with no owner keeps producing the same error: next week's claim from the same list will fail the same way, and nobody is charged with fixing the list.

Dates as matching keys

When two systems share no identifier, they match on attributes, and a date of birth is the usual one. It is a poor key on its own and a dangerous one when its order is not stated. The text 03/04/1990 is 3 April 1990 to a reader that expects day first and 4 March 1990 to a reader that expects month first. Both readings are valid dates, so no range check fails. Matching on attributes is a fallback with rules of its own, and those rules belong in the contract too; it is not the exact key match this chapter asks for. ISO 8601-1:2019 removes the ambiguity by ordering the elements from largest to smallest, year, month, day: 1990-04-03.

Hop 4, the date of birth. The guest entered her date of birth on the group's EU booking site, whose form uses day, month, year: 03/04/1990, meaning 3 April. The hotel's guest profile stored it as text, as typed. Claim intake parsed the text as month, day, year and stored 4 March 1990. The hospital's record held 3 April. When intake tried to link its claimant to the hospital's patient, the date of birth disagreed. It was the second matching key to fail in the claim; the name had already failed at hop 3.

THE HOP · 4 · 2025 · EU BOOKING SITE (DD/MM/YYYY) TO HOTEL GUEST PROFILE, THEN CLAIM INTAKE (MM/DD/YYYY)

Record	Date of birth (One Night, Every Value, illustrative), hop 4, 2025
From	EU booking site (DD/MM/YYYY)
To	Hotel guest profile, then claim intake (MM/DD/YYYY)
Mechanism and schedule	The date of birth. Entered as text on the EU form, stored as text in the profile, parsed at claim intake
Format change; what was lost or assumed	IDENTITY '03/04/1990' entered as 3 April; parsed as 4 March. **The second matching key fails: the claim intake record and the hospital record now disagree on the date of birth**
Accountable owner	Group guest data team
Who can change it	The guest (on the form); claim intake staff
Classification; retention or erasure duty	Personal data
Read by (people, apps, models, agents)	Claim intake, identity matching, the triage agent
Cost per month; basis	One-off: part of the same manual identity check (not monthly) (illustrative)

Notice what did not happen. Nothing was corrupted in storage, and the profile kept exactly what the guest typed. The meaning was lost at the parse, because the text did not carry its order and the parser assumed one. The fix is the same as for the policy number: the boundary declares the form (ISO 8601, year first) and the parser refuses anything else instead of guessing. A date of birth used for matching is also personal data, so the contract that names its form also names who may receive it.

Check digits: what they catch and what they miss

A **check digit** is a character computed from the other characters of a key and appended to it, so that a reader can tell whether the key was copied correctly without looking it up. The best known scheme is older than most databases. Hans Peter Luhn of IBM filed for US patent 2,950,048, "Computer for Verifying Numbers", on 6 January 1954; it was granted on 23 August 1960 and describes a hand device for computing a check digit and for verifying "numbers which already have a check digit appended". The Luhn formula is now Annex B of ISO/IEC 7812-1 (fifth edition, 2017), the numbering standard for identification card issuers, and the NPI uses it, computed as if the NPI were preceded by the prefix 80840 (80 for health applications, 840 for the United States).

The CMS statement of the formula is three steps: double every second digit starting from the rightmost, add the digits of the products to the untouched digits, and subtract the total from the next multiple of ten. On the invented body 00417730:

```
Body:                 0 0 4 1 7 7 3 0
Read from the right: 0 3 7 7 1 4 0 0
Double the 1st, 3rd, 5th, 7th: 0 to 0, 7 to 14, 1 to 2, 0 to 0
Sum of digits: 0 + 3 + (1 + 4) + 7 + 2 + 4 + 0 + 0 = 21
Check digit:   30 - 21 = 9
```

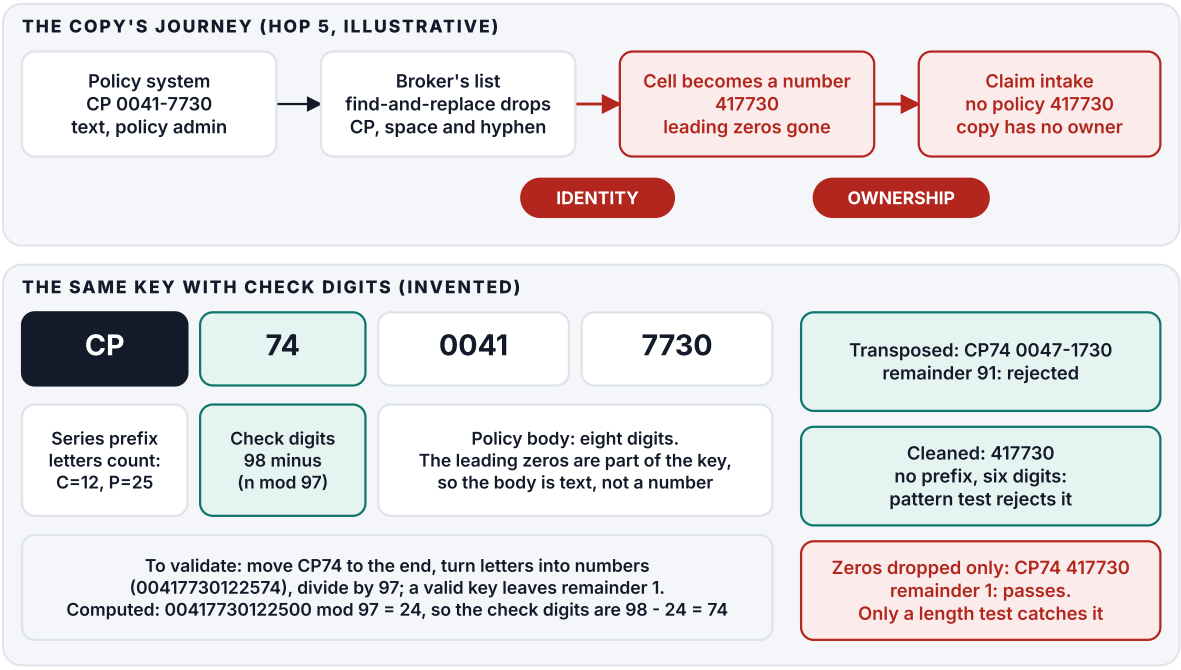
The International Bank Account Number uses a different scheme, with two check digits. ISO 13616, first published in 1997 and now ISO 13616-1:2020, computes two check digits with ISO 7064 MOD 97-10: move the country code and "00" to the end, turn letters into numbers (A is 10, B is 11 and so on), divide the whole string by 97, and subtract the remainder from 98. A valid IBAN leaves a remainder of 1. Applied to the invented key, with the series prefix playing the part of the country code:

```
Key without check digits:   CP .. 00417730
Rearranged:                 00417730 CP 00
Letters to numbers:         00417730 12 25 00
00417730122500 mod 97 = 24  so check = 98 - 24 = 74
Published key:              CP74 0041-7730
Validate: 00417730122574 mod 97 = 1
```

Now test both schemes against two copying errors and against hop 5. A typing slip that swaps two neighbouring digits, CP74 0047-1730, is caught: it leaves a remainder of 91, and the Luhn digit for 00471730 is 2, not 9. The cleaned value 417730 of hop 5 is caught too, but only because it no longer has the prefix or the length; it fails the pattern, not the arithmetic. And a copy that drops only the zeros, CP74 417730, passes MOD 97-10 with a remainder of 1, and the Luhn digit of 417730 is 9, the same as for 00417730. Leading zeros change neither calculation. Figure 7.1 puts the journey and the anatomy side by side.

FIGURE 7.1

A key's life and its check digit: the copy that lost its zeros, and what a check digit would and would not have caught



Top: hop 5, the policy number as it moved to the broker's list and back to claim intake. Bottom: the same invented key with IBAN-style check digits; the check catches a transposition of two digits but not lost leading zeros, which only a length and pattern test catches.

Author's diagram based on the check-digit procedure of ISO 13616 as first published (ISO 13616:1997, clause 6 and Annex B; ISO 7064 MOD 97-10; the current edition is ISO 13616-1:2020) and the CMS statement of the Luhn formula (2004); key and check digits invented and computed for this book.

The lesson is precise. A check digit protects the characters it covers against copying errors. It does not protect the type or the length, and it says nothing about whether the key exists. The NPI pairs its check with a fixed length of ten digits, and a company that adds check digits to its own keys should do the same. A contract for any identifier states the pattern, the length and the check together, and tests all three.

UUIDs: unique, not secret, not always text

A **universally unique identifier** (UUID) is a 128-bit value that many systems can generate independently; in the words of RFC 9562, a UUID "is 128 bits long and requires no central registration process". RFC 9562 (May 2024) defines it and obsoletes RFC 4122. It describes versions 1 to 8; version 4 is random, and version 7 is built from a Unix-epoch timestamp in milliseconds plus random bits, so that values sort roughly in creation order.

The RFC says "Implementations SHOULD utilize UUIDv7 instead of UUIDv1 and UUIDv6 if possible" (section 5.7) and notes that "Time-ordered monotonic UUIDs benefit from greater database-index locality" (section 6.11).

Three properties belong in the contract. First, representation: the familiar text form is 36 characters, hexadecimal in groups with hyphens, and section 4 allows letters in "all uppercase, all lowercase, or mixed case", so text comparison must normalise case first. Section 6.13 adds that storing the text form takes 288 bits for a 128-bit value and that, "where feasible, UUIDs SHOULD be stored within database applications as the underlying 128-bit binary value". Second, meaning: a version 7 UUID carries its creation time, and section 8 notes that the timestamp signals the order of creation. Third, secrecy: "Implementations SHOULD NOT assume that UUIDs are hard to guess", and UUIDs "MUST NOT be used as security capabilities" (section 8). A link that grants access because the identifier in it is long is an access control written in the wrong place.

SOURCED FAILURE: GENE SYMBOLS READ AS DATES

In *Genome Biology* (volume 17, article 177, 23 August 2016), Mark Ziemann, Yotam Eren and Assam El-Osta reported what happened to gene identifiers in the supplementary files of published papers. They screened files from 18 journals published between 2005 and 2015: 35,175 Excel files, holding 7,467 gene lists attached to 3,597 papers. They confirmed gene name errors in 987 files from 704 articles, and estimated that 19.6 % of articles with gene lists in Excel files were affected. The mechanism is a reader guessing a type. Gene symbols such as SEPT2 and MARCH1 "are converted by default to '2-Sep' and '1-Mar'", and, as an earlier report they cite had shown, identifiers such as the RIKEN accession 2310009E13 were converted to the floating-point number 2.31E+13. The symbols were identifiers; the software read them as a date and a number.

The fix came from the owner of the code table, not from the software. In August 2020 the HUGO Gene Nomenclature Committee published new guidelines (Bruford et al., *Nature Genetics* 52, 754 to 758). Box 3 records that "all symbols that auto-converted to dates in Microsoft Excel have been changed": "SEPT1 is now SEPTIN1" and "MARCH1 is now MARCHF1". An authority renamed part of a vocabulary so that its identifiers would survive careless readers. Every renamed symbol is now a version question for any dataset that joins old and new names, which is Chapter Eight's subject.

TYPE CONTRACT: IDENTIFIERS (THE COMMERCIAL POLICY NUMBER)

1. Type, precision and rounding. A string, never a number. Canonical form: "CP", then two check digits where the series adds them, then exactly eight digits (CP00417730); display form CP 0041-7730. No arithmetic, so no rounding. *Test:*

the value matches the pattern and length; a numeric type or a value of any other length is held.

2. Encoding and serialization. ASCII letters and digits within UTF-8; upper case; separators removed for matching and added only for display. *Test:* matching compares canonical forms; a value that changes under canonicalisation other than by removing the space and hyphen is held.

3. Unit or currency. Not applicable. *Test:* no unit or amount is ever derived from the key; a schema that declares the column numeric fails review.

4. Time basis. Valid from issue; never reissued to another policy. Lookups resolve the policy as in force on the date of loss. *Test:* intake retrieves the policy version for the date of loss, not today's.

5. Identity scope. The insurer's commercial lines series (CP), separate from the A- and B- series; issued by policy administration. Where check digits are added, they are verified on arrival. *Test:* the prefix is in the list of live series and the key resolves in the issuer's register.

6. Absence. Missing means not yet known; never zero, never an all-zero body, never the empty string. *Test:* CP00000000, 0 and "" are rejected, not stored.

7. Authority. Policy administration owns the register. Every copy (a broker's list included) names its source and its owner; brokers of record may receive the key. *Test:* intake refuses a list whose owner field is empty, and returns a failed key to that owner.

ERA CARD

Reader. The register clerk read a line number, the punched card a fixed field, the database a surrogate, the web API a UUID; each boundary between them needed the natural key and its form.

Security. Identifiers are not secrets: RFC 9562 says UUIDs must not be used as security capabilities. A date of birth used for matching is personal data, and every copy that carries it widens who holds it.

Cost. A rejected key costs minutes at the boundary; a key that fails to resolve, as at hop 5, cost a two-day delay and a list that keeps failing; a wrong key that resolves would attach the claim to someone else's policy (illustrative).

AI. A model sees 417730 as a plausible number and 03/04/1990 as a plausible date. Asked to find the policy, it may offer the nearest match. The contract tells the agent to match canonical forms exactly and to report a key that fails its pattern.

Monday rule: declare every identifier column that crosses a boundary as text with a pattern and a length, and make intake reject a key that fails them instead of searching for something close.

CHAPTER EIGHT

Codes, Units and Enumerations



The insurer's claim summary for the guest carried two clinical values. One was a laboratory result, "COHb 12 %". The other was a diagnosis code from the hospital's bill. Both matched what the hospital had sent, character for character, and both had become harder to read than the originals. The result had lost the reference range, the time of the blood draw and a note that oxygen had been given before it. The code had lost the name of its code system and the edition it came from. A code, a unit or an item from a list is a value whose meaning lives somewhere else: in a table, a standard or a habit. When the value travels and the table does not, the reader supplies one. The decision this chapter changes: **every coded value travels with its code system and version, every quantity with a unit from a named unit system, and every enumeration with its list of values and its owner**.

Code tables and their owners

A **code** is a short value that stands for a longer meaning defined in a **code table**: a list of codes, each with its meaning, maintained by someone. *From Tally to Token* told how a status code 7 reached a warehouse without the table that said what 7 meant; that story is not repeated here. The general point is that a code has three properties a plain value does not have. It has an owner, who may add, retire or redefine codes. It has versions, because the owner changes the table over time. And it has a scope: the same two letters can mean different things in two tables, or in one table in two decades.

Country codes show all three. ISO 3166-1 (fourth edition, 2020) is maintained by the ISO 3166 Maintenance Agency, and the codes change when countries do. Some changes touch only the label. In a change effective 11 July 2022, the English short name for the code TR became "Türkiye" in place of "Turkey"; the code TR did not change. A system that stores the code and looks up the name keeps working. A system that stores the English name as text now holds two spellings of one country, and a report that groups by name splits it in two.

Other changes touch the code itself. Czechoslovakia divided in 1993, and ISO's glossary gives its formerly used code element as CSHH. The two letters CS were later used again: the IANA language subtag registry, which tracks ISO 3166-1, lists CS as "Serbia and Montenegro", deprecated in October 2006 with the note "see RS for Serbia or ME for Montenegro". A field that holds CS from a 1990 record and CS from a 2005 record holds two countries. ISO now states that once a two-letter code element "is assigned and then deleted, it cannot be re-used for 50 years". Fifty years is long, but it is a period, not a promise of permanence, and a dataset that keeps records for a century needs the date of the code as well as the code.

Controlled vocabularies and their editions

Health care runs on three large code systems, each with its own owner and release rhythm. **ICD-10-CM**, the US clinical modification of the World Health Organization's ICD-10, is maintained in the United States by the National Center for Health Statistics at CDC. **LOINC**, published by the Regenstrief Institute, names laboratory tests and other observations; version 2.83 was released on 19 August 2026. **SNOMED CT**, maintained by SNOMED International, is a clinical terminology whose International Edition is now released every month.

ICD-10-CM changes every year, and the changes are not only additions. CDC's files say that FY2026 codes "should be used for healthcare services provided from October 1, 2025, through September 30, 2026"; an update followed on 1 April 2026, and the FY2027 codes took effect on 1 October 2026. CDC's addenda for FY2026 count 74,719 codes against 74,260 the year before, with 487 additions, 28 deletions and 38 revisions. Two examples from those files show why the edition matters:

- **A code whose meaning narrowed.** L02.212 read "Cutaneous abscess of back [any part, except buttock]" in FY2025. In FY2026 it reads "Cutaneous abscess of back [any part, except buttock and flank]", and a new code, L02.217, "Cutaneous abscess of flank", takes the cases it no longer covers. The same five characters mean slightly less after 1 October 2025.
- **A code that stopped existing.** E78.01 "Familial hypercholesterolemia" was deleted in FY2026 and replaced by three more specific codes, E78.010, E78.011 and E78.019. A service dated September 2025 may validly be coded E78.01; a service dated November 2025 may not.

HL7's FHIR standard has a place for exactly this. In FHIR R4 (version 4.0.1, October 2019) a code travels as a `Coding` with a `system`, a `code` and an optional `version`, which the specification defines as "The version of the code system which was used when choosing this code". It adds that a well-maintained code system keeps meanings consistent across versions, but "this cannot consistently be assured, and when the meaning is not guaranteed

to be consistent, the version SHOULD be exchanged". The system for ICD-10-CM is the URI `http://hl7.org/fhir/sid/icd-10-cm`, and FHIR requires ICD codes to be written with the period included. A code stored as L02212 in one place and L02.212 in another is the identifier problem of Chapter Seven again.

Hop 8, the diagnosis code. The hospital's coder assigned the guest's diagnosis codes from ICD-10-CM, in the edition in force on the date of service: the FY2026 codes, in force since 1 October 2025. The itemised bill printed the codes. The insurer's claim system stored them in a plain text column, with no column for the code system and none for the edition. Nothing was wrong on the day it was stored. A year later, with FY2027 in force, a reader of that column cannot tell which code system gave the code its meaning, or which edition. For one claim the cost is small. For claims analytics that join codes across years it is large, because every revision and deletion in the annual addenda silently changes what a count by code means (illustrative).

THE HOP · 8 · 2025 · HOSPITAL CODER (ICD-10-CM, EDITION IN FORCE ON THE DATE OF SERVICE) TO ITEMISED BILL, THEN THE CLAIM SYSTEM

Record	Diagnosis code (One Night, Every Value, illustrative), hop 8, 2025
From	Hospital coder (ICD-10-CM, edition in force on the date of service)
To	Itemised bill, then the claim system
Mechanism and schedule	The diagnosis code. The bill prints codes; the claim system stores them in a text column
Format change; what was lost or assumed	MEANING The code is stored without its code system or edition. **A later reader cannot tell which code system and which edition gave the code its meaning**
Accountable owner	Hospital coding (source); insurer claims operations (stored code)
Who can change it	Hospital coders; claims staff
Classification; retention or erasure duty	Protected health information
Read by (people, apps, models, agents)	The nurse reviewer, claims analytics, the triage agent
Cost per month; basis	Monthly: small per claim; larger for analytics that join codes across years (illustrative)

Units: the number is half the value

A quantity is a number and a unit, and only the pair has meaning. The **Unified Code for Units of Measure** (UCUM; version 2.2, June 2024) exists for the case where machines exchange quantities. It describes itself as "a code system intended to include all units of measures being contemporarily used in international science, engineering, and business", whose purpose "is to facilitate unambiguous electronic communication of quantities together with their units". UCUM gives each unit a code with a defined grammar: `%` is percent, defined as 10^{-2}; `[degF]` is degree Fahrenheit; `mg/dL` is built from prefix, unit and division. Its tables list a case-sensitive and a case-insensitive form for each unit, and the two are not interchangeable. FHIR names UCUM with the URI `http://unitsofmeasure.org` and marks its codes as case-sensitive.

A unit in a column heading, a report title or a habit is a unit that does not travel. *From Tally to Token* told the case of Mars Climate Orbiter, lost in 1999 because a ground software file reported impulse in pound-seconds where the interface required newton-seconds. The case below is older, smaller and closer to the desk.

SOURCED FAILURE: AIR CANADA FLIGHT 143, 23 JULY 1983

The *Final Report of the Board of Inquiry* (Commissioner Justice George H. Lockwood, April 1985) records that Air Canada's Boeing 767 C-GAUN, flying as AC 143 from Montreal to Edmonton with a stop in Ottawa, "ran out of fuel over Red Lake, Ontario, about halfway to its destination" and made an emergency landing at a disused military airbase at Gimli, Manitoba. There were two pilots, 6 cabin crew and 61 passengers; "there was no loss of life, nor were there any significant physical injuries" (Part I, p. 9). The aircraft's fuel gauges were blank, so the fuel was measured by drip stick in litres, and the flight plan required fuel in kilograms. The factor used to convert, 1.77, converts litres to *pounds*; the Board notes that the factor for litres to kilograms "is typically around .8" (Part II, p. 28), and the operations manual gave .80 for Jet A (Part III, p. 41).

The Board reconstructs the Montreal calculation (pp. 41 to 42): 7,682 litres on board times 1.77 gave 13,597, "in pounds but everyone involved believed it to be kilograms". Subtracted from the minimum 22,300 kilograms, that left 8,703, divided by 1.77 to give 4,916 litres to add, rounded to 5,000. The Board's correct calculation keeps the same factor and converts the pounds to kilograms by dividing by 2.2: 6,180 kilograms on board, 16,120 kilograms to add, and 20,036 litres, "in the area of 20,000 litres": four times what was added. The same mistake was made again in Ottawa. The Board's preface names the setting: the introduction of "a so-called metric aircraft into a non-metric fleet". Every number in the calculation was arithmetically right. The units were carried in people's heads, and two heads held different ones.

The enterprise version is quieter and more frequent: a weight column that changed from pounds to kilograms when a supplier changed, a currency field that defaults to the company's home currency, a percentage stored as 12 in one table and 0.12 in another. In each case the fix is the same: the unit is a field of its own, filled from a named unit system, and a reader refuses a quantity whose unit it does not recognise.

Enumerations and booleans

An **enumeration** is a field restricted to a short list of values: a claim status, a room type, a yes or no. It is a small code table, usually without an owner, a version or a document, which is why it fails in the same ways at a smaller scale. Three questions belong in its contract. What does each value mean? What does a blank mean? What happens when a value outside the list arrives?

The boolean is the smallest enumeration and the most misunderstood. A boolean column that allows null has three states, not two. PostgreSQL's documentation says its boolean type "can have several states: 'true', 'false', and a third state, 'unknown', which is represented by the SQL null value". It accepts as input not only true and false but yes and no, on and off, 1 and 0, and unique prefixes such as t and n. Files exported from older systems add their own forms: Y, N, a space, an empty string. A field such as "smoke detector tested: Y/N/blank" therefore has at least three values, and the third is the one that matters most. Chapter Ten takes absence in depth; here the rule is that an enumeration lists its absence value explicitly, as one of its values, with a meaning: not asked, not known, not applicable.

The values outside the list are the second risk. When a source adds a status, a reader can reject the record, map it to a catch-all such as "other", or pass it through as text. Each choice is reasonable and each must be written down, because a silent catch-all turns a new status into an old one. Chapter Twelve shows which schema changes break which readers; for an enumeration, adding a value is the change most often treated as harmless.

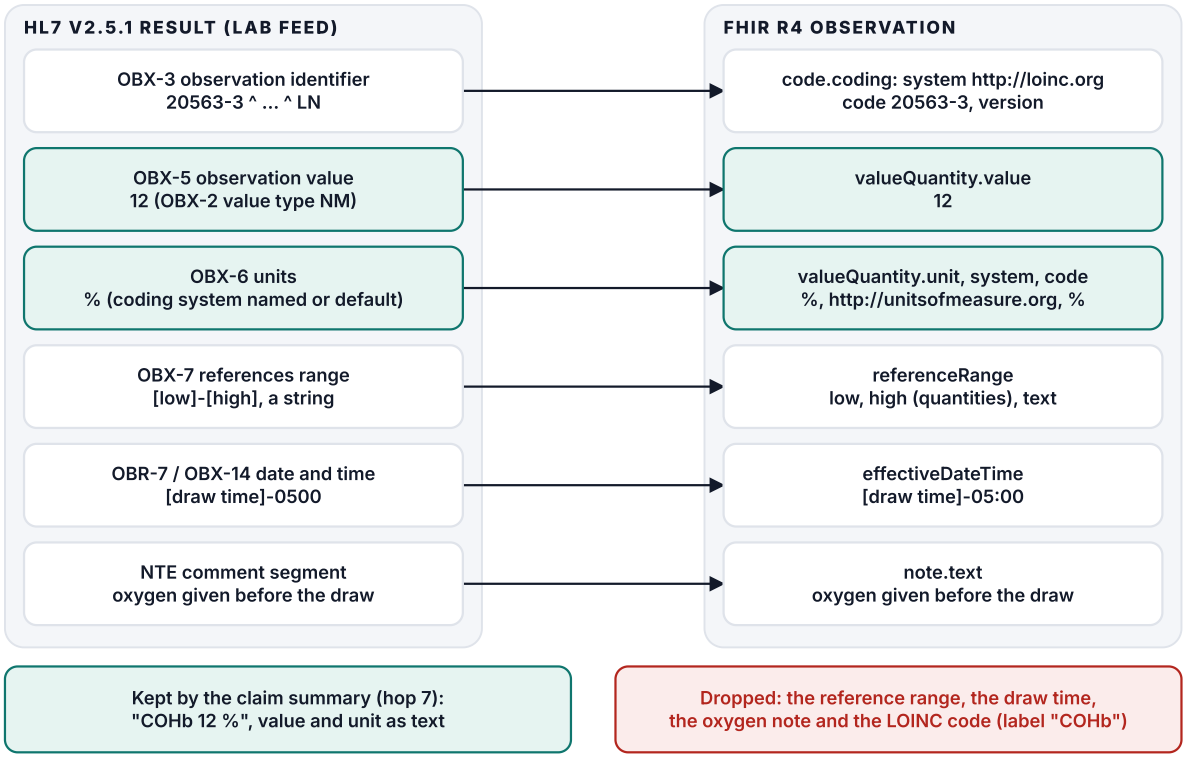
A result and its context: OBX and the FHIR Observation

A laboratory result is the clearest case of a value that is not one value. In HL7 version 2.5.1, chapter 7, a result travels in an OBX segment, one field after another. OBX-3 is the observation identifier, here a LOINC code. OBX-5 is the observation value, "the value observed by the observation producer", whose data type is named in OBX-2. OBX-6 is the units; in version 2.5.1 the default coding system is ISO+, and "only ISO+ abbreviations need be sent", so the unit system is often implicit. OBX-7 is the reference range, a string such as "lower limit-upper limit". The time that matters clinically is in OBR-7, "the clinically relevant date/time of the observation", or OBX-14 when results under one order differ. Comments travel in separate NTE segments.

FHIR R4 gives each of those parts a named home in an `Observation` resource: `code` (a CodeableConcept whose Codings carry system and version), `valueQuantity` (value, unit, unit system and unit code), `referenceRange` (low, high and text), `effectiveDateTime`, and `note`. Figure 8.1 sets the two side by side for an invented carboxyhaemoglobin result. The LOINC code is real: 20563-3, "Carboxyhemoglobin/Hemoglobin.total in Blood", a mass fraction whose example unit is %.

FIGURE 8.1

An OBX segment beside its FHIR Observation: every part of the result has a home, and the claim summary kept two of them



The carboxyhaemoglobin result as a laboratory feed and as a FHIR resource carry six matching parts (an illustrative mapping; implementations may map the draw time to a Specimen resource instead); the PDF extraction at hop 7 kept the value and the unit and dropped the rest.

Author's diagram based on HL7 Version 2.5.1 chapter 7 (section 7.4.2) and HL7 FHIR R4 (4.0.1) Observation; LOINC 20563-3 from loinc.org; values invented, bracketed items left as placeholders.

Why the parts matter is written into LOINC itself. Its description of carboxyhaemoglobin says that a normal individual has about 1 or 2 % of haemoglobin in that form, up to 5 % depending on environmental circumstances, and that smokers can have levels as high as 10

% or higher. A reader can see that 12 % is above the usual range. Without the reference range, the draw time and the note about oxygen therapy, no reader, person or model, can say what the exposure was.

Hop 7, the carboxyhaemoglobin result. The hospital laboratory's analyser sent the result over HL7 v2 with value, unit and reference range, and the hospital's records showed the time of the draw and a note that oxygen had been given before it. The records reached the insurer as PDF documents. The insurer's extraction service kept the result line, "COHb 12 %", and dropped the notes around it: the reference range, the specimen time and "oxygen given before the draw". The nurse reviewer and the triage agent received a number that could not be judged. Without the specimen time and the oxygen note, nobody could say what the exposure had been; getting them back took a records request to the hospital, about a week (illustrative).

THE HOP · 7 · 2025 · LABORATORY ANALYSER, HL7 V2 OBX (VALUE, UNIT, REFERENCE RANGE) TO PDF RECORDS, THEN THE CLAIM SUMMARY

Record	Carboxyhaemoglobin result (One Night, Every Value, illustrative), hop 7, 2025
From	Laboratory analyser, HL7 v2 OBX (value, unit, reference range)
To	PDF records, then the claim summary
Mechanism and schedule	The carboxyhaemoglobin (COHb) result. The extraction service keeps the result line and drops the notes around it
Format change; what was lost or assumed	CONTEXT 'COHb 12 %' kept; the reference range, the specimen time and 'oxygen given before the draw' dropped. **Without the specimen time and the oxygen note, a reader cannot judge the exposure from the number**
Accountable owner	Hospital laboratory (source); insurer claims operations (summary)
Who can change it	Laboratory staff; the extraction service
Classification; retention or erasure duty	Protected health information
Read by (people, apps, models, agents)	Clinicians; the nurse reviewer; the triage agent
Cost per month; basis	One-off: a records request to the hospital, about a week (not monthly) (illustrative)

No field was corrupted at hop 7, and no code system was misread. The loss is CONTEXT: the value was kept and the values that interpret it were not. That is the pattern of this chapter. A code needs its table and edition, a quantity needs its unit, a result needs its range and time, and each of those is a field of its own that a copy can drop.

TYPE CONTRACT: CODES AND UNITS (THE DIAGNOSIS CODE AND THE COHB RESULT)

1. Type, precision and rounding. A code is a string from its code system, never a number; a result is a decimal with the precision the laboratory reported, not rounded on copy. *Test:* the code matches the code system's pattern (ICD-10-CM with the period); the result keeps its reported digits from source to summary.

2. Encoding and serialization. FHIR Coding (system, version, code, display) and Quantity (value, unit, system, code), or their columns in a table; never a single text field such as "COHb 12 %". *Test:* each part is a separate field; a value that arrives as one string is held.

3. Unit or currency. UCUM, by code: `%` for the result. Codes have no unit. *Test:* the unit code parses as UCUM and is one the field allows; a result without a unit is held.

4. Time basis. The code's edition is the one in force on the date of service; the result carries its specimen time as an instant with offset. *Test:* the edition matches the date of service (FY2026 for 2 November 2025); a result without a specimen time is held.

5. Identity scope. The code system is named by URI (`http://hl7.org/fhir/sid/icd-10-cm`, `http://loinc.org`); a result belongs to one patient, one specimen and one order. *Test:* the system URI is present and on the allowed list; the result links to its specimen.

6. Absence. No code means not coded, never a default code; an empty reference range means the laboratory sent none, and the field says so. *Test:* placeholder codes and empty ranges without a reason are rejected.

7. Authority. The code system's owner (NCHS for ICD-10-CM, Regenstrief for LOINC) defines meaning; the hospital laboratory owns the result; the patient's authorisation defines who may receive it. *Test:* the code is valid in the stated edition's code file; the recipient is within the authorisation's scope.

OUR VIEW, AS OF NOVEMBER 2026

Code systems will keep changing every year, and that is healthy. The cost falls on companies that store codes as bare strings, because every edition turns some of their history into a different vocabulary. Storing the system and edition costs two short columns. Reconstructing them later costs a project.

ERA CARD

Reader. The clerk knew the codes by heart, the card shop kept them on a sheet, the relational era in a lookup table, the web era in a URI; a model knows only what is sent with the value.

Security. A diagnosis code and a laboratory result are protected health information; a context field such as a clinical note can hold more than the authorisation covers, so each part's recipients are named in the contract.

Cost. Two columns for system and edition per coded field, and one for the unit, against a week-long records request per claim and analytics that drift every October (illustrative).

AI. A model shown "COHb 12 %" will place it on a scale it learned from text, which is the reader's guess the contract exists to prevent. Given the range, the draw time and the note, it can reason; without them it should say the value cannot be judged.

Monday rule: find every column that stores a code without its code system and edition, or a quantity without its unit, and add those fields before the next annual code release lands.

CHAPTER NINE

Place



In August 2016 the IETF published RFC 7946, the GeoJSON format, and settled one question for every file that follows it: a position is written "longitude and latitude … precisely in that order". The registry entry most systems cite for WGS 84 coordinates, EPSG:4326, lists its axes the other way round, latitude first. Both are correct, and a pair of numbers copied from one to the other is wrong without any error. In the days after the fire the insurer's catastrophe team built its claims risk map from the hotel's property schedule, and the hotel moved about 14,800 kilometres, to Antarctica. The decision this chapter changes: **a coordinate pair never travels without the name of its reference system, and that name decides which number is the latitude**.

A place is two numbers and a datum

A **coordinate** is a number measured from an origin along an axis. A pair of them names a place only when the reader knows the **datum**, the model of the Earth the numbers are measured on. The datum behind almost every business coordinate is the World Geodetic System 1984. The National Geospatial-Intelligence Agency, which maintains it, describes WGS 84 as "an Earth-centered, Earth-fixed terrestrial reference system and geodetic datum", the standard US Department of Defense definition of a global reference system for geospatial information, and "the reference system for the Global Positioning System". Its ellipsoid has a semi-major axis of 6,378,137 metres and a flattening of 1/298.257223563.

WGS 84 is not one fixed frame. The EPSG Geodetic Parameter Registry, maintained by the International Association of Oil and Gas Producers (IOGP), records the datum of EPSG:4326 as an **ensemble** of eight realizations, from the original Transit frame to G2296, with an ensemble accuracy of 2 metres. For an insured building, two metres does not matter. For a pipeline, a property boundary or a drone landing pad it can, and then the contract names the realization and the epoch, not just "WGS 84".

Which number comes first

The EPSG registry defines CRS 4326 with coordinate system 6422: "Axes: latitude, longitude. Orientations: north, east." Latitude is the first number. The Open Geospatial Consortium (OGC) defines a second identifier for the same datum, CRS84, named "WGS 84 longitude-latitude", whose first axis is longitude. GeoJSON adopted it. RFC 7946, section 4, says GeoJSON coordinates use WGS 84 "with longitude and latitude units of decimal degrees", equivalent to OGC's CRS84, and it removed the 2008 specification's option to name other systems because that option "has proven to have interoperability issues".

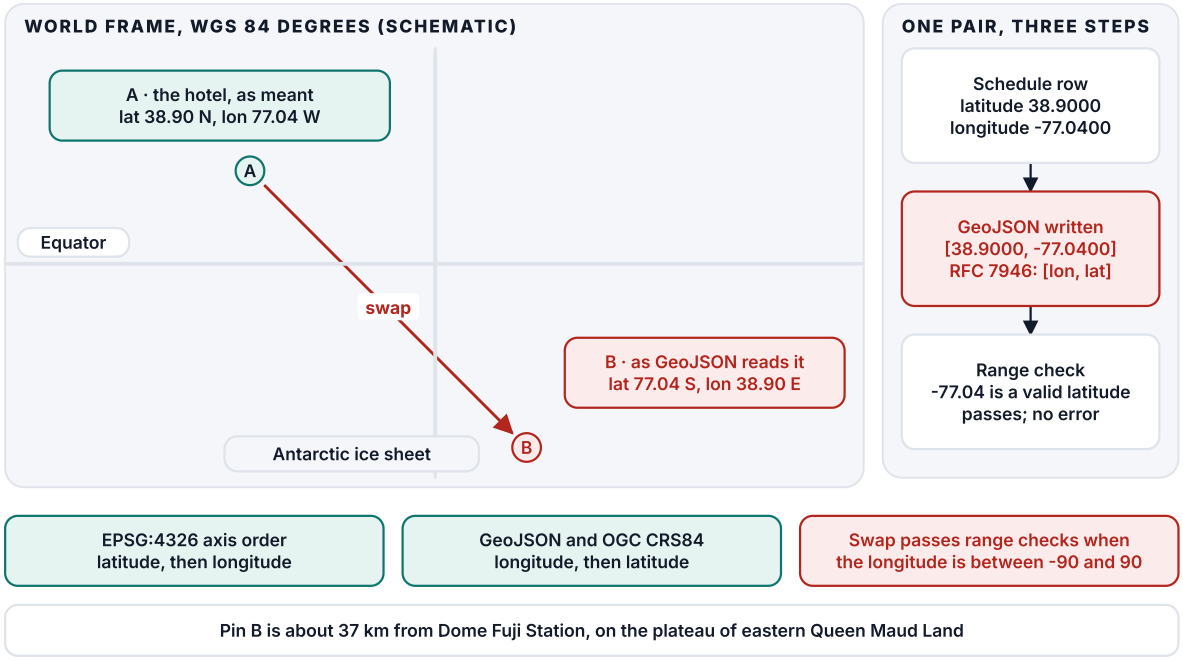
So the same datum has two common axis orders, and both are standard. People say "lat, long" when they speak; mapping libraries and web formats often write x before y, which for a geographic system means longitude first. Nothing in the numbers says which order a file used. The order lives in the identifier, if anyone wrote it down, or in the column names, if the file has them.

The range check that most pipelines rely on is weaker than it looks. A latitude must lie between -90 and 90; a longitude between -180 and 180. A swapped pair fails that check only when the true longitude is beyond 90 degrees east or west. For anywhere between those meridians, which takes in all of Europe, Africa and South America, western Asia and the eastern United States, both orders pass, and the error lands somewhere real.

The Apache Parquet format now carries this rule in the file: its GEOMETRY and GEOGRAPHY logical types take an optional crs parameter that, if unset, defaults to OGC:CRS84, "which means that the geometries must be stored in longitude, latitude based on the WGS84 datum".

FIGURE 9.1

Two pins: the same pair of numbers, read in two orders, lands 14,844 km apart



The schedule meant latitude 38.9000, longitude -77.0400 (pin A); written into GeoJSON in the same order, a reader takes 38.9000 as the longitude and -77.0400 as the latitude (pin B), and both numbers pass a latitude range check; the distance is a great-circle distance on a spherical Earth.

Author's diagram based on the EPSG registry (EPSG:4326 axes latitude, longitude), RFC 7946 sections 3.1.1 and 4, and the OGC CRS84 definition; positions and the 14,844 km great-circle distance computed for this book; place of pin B from the SCAR Composite Gazetteer of Antarctica (Queen Maud Land; Dome Fuji Station). Schematic, not a map.

Digits are not accuracy

Decimal degrees look exact, and the number of digits is easy to mistake for how well the place is known. RFC 7946 rules that out: "the number of digits of the values in coordinate positions MUST NOT be interpreted as an indication to the level of uncertainty". The digits do set a resolution. Computed for this book on the WGS 84 ellipsoid, one degree of latitude near 38.9 degrees north is about 111.0 kilometres, and one degree of longitude about 86.7 kilometres. The hotel's four decimal places, 0.0001 degree, resolve about 11 metres north to south and 9 metres east to west. Two decimal places resolve about a kilometre. The RFC's own note is that six decimal places "amounts to about 10 centimeters".

Resolution is not accuracy. A point geocoded from a street address to six decimals may still sit at the middle of a parcel, on the street in front, or at the centre of a postal code. The contract therefore states two things the digits cannot: what the point stands for (building centroid, main entrance, parcel) and how it was obtained (survey, GPS fix, geocoder and version).

Addresses: text or structure

An **address** is the other way a place travels, and it fails differently. As one line of text it keeps what the writer typed and gives the reader the job of splitting it. As structure it names its parts. The US Postal Service's Publication 28, Postal Addressing Standards (October 2024 edition), defines a standardized address as "one that includes all required address elements" using the Postal Service abbreviations, with the delivery line and last line "validated with the ZIP+4 file and City State file". Other countries order the same parts differently, and a parser written for one country's order splits another's wrongly. The payment message of hop 11 shows the cost of forcing free text into structured fields; Chapter Thirteen tells it.

A coordinate and an address are two values for one place, and they drift apart. The useful test is to keep both and check one against the other: the point must fall inside the country, state or district, or postal area the address names. That single check catches this swap, and most swaps of a pair that every range check passes; it misses only a swap that lands inside the same area.

TYPE CONTRACT: PLACE (THE HOTEL LOCATION OF HOP 9)

1. Type, precision and rounding. A geographic point in decimal degrees: latitude from -90 to 90, longitude from -180 to 180, at most six decimals; the point stands for the building centroid. *Test:* a value out of range, or with more than six decimals, is held.

2. Encoding and serialization. In tables, two named columns, latitude and longitude, never an unnamed pair; in GeoJSON, [longitude, latitude] per RFC 7946. *Test:* every writer records the CRS identifier it wrote, and a GeoJSON writer reads the latitude and longitude columns by name, never by position.

3. Unit or currency. Degrees for both axes; a height, if present, in metres above the WGS 84 ellipsoid. *Test:* a file in projected metres (eastings and northings) under a geographic identifier is held.

4. Time basis. The date the location was verified; for metre-level work, the realization and epoch (the WGS 84 ensemble is good to about 2 metres). *Test:* a point without a verification date is flagged for review before it feeds an accumulation report.

5. Identity scope. The insured building at the schedule's address, policy CP 0041-7730; not the parcel, not the postal code centre. *Test:* the point falls inside the state or district and the postal area of the schedule's address.

6. Absence. Missing means not yet geocoded; never written as (0, 0) or as a default point. *Test:* a point at (0, 0), or outside every land polygon for a land risk, is held as missing.

7. Authority. Underwriting owns the schedule; the catastrophe modelling team owns the map script; the CRS is named as EPSG:4326 (latitude first) in the schedule and OGC:CRS84 in the map; the geocoder and its version are recorded. *Test:* the map build fails when a source has no CRS identifier.

THE HOP · 9 · 2025 · PROPERTY SCHEDULE OF THE POLICY (LATITUDE, LONGITUDE COLUMNS) TO CLAIMS RISK MAP (GEOJSON, LONGITUDE FIRST)

Record	Hotel location (One Night, Every Value, illustrative), hop 9, 2025
From	Property schedule of the policy (latitude, longitude columns)
To	Claims risk map (GeoJSON, longitude first)
Mechanism and schedule	The hotel's location. A script builds GeoJSON points from the schedule's two columns in the order they appear
Format change; what was lost or assumed	MEANING (38.9000, -77.0400) written as [38.9000, -77.0400] where GeoJSON expects [longitude, latitude]. **The point lands at latitude -77.04, longitude 38.90, on the Antarctic ice sheet; it passes range checks because both numbers are valid latitudes**
Accountable owner	Insurer underwriting (schedule); catastrophe modelling team (map)
Who can change it	Underwriting assistants; the map script's owner
Classification; retention or erasure duty	Commercial record
Read by (people, apps, models, agents)	Catastrophe modellers, the adjuster's map view
Cost per month; basis	One-off: a hotel missing from the accumulation report until someone notices (not monthly) (illustrative)

Hop 9 is the map. The property schedule of the hotel's policy listed the location in two columns, latitude 38.9000 and longitude -77.0400, latitude first as people write it. The catastrophe modelling team's script built GeoJSON points from the schedule's columns in the order they appeared, so it wrote [38.9000, -77.0400]. A GeoJSON reader takes the first number as the longitude. It placed the hotel at latitude -77.04, longitude 38.90.

Nothing failed. Both numbers are valid latitudes, and 38.90 is a valid longitude, so every range check passed. The point is real: computed for this book as a great-circle distance on a spherical Earth, it lies about 14,844 kilometres from the hotel, on the Antarctic ice sheet, on the high plateau of eastern Queen Maud Land, about 37 kilometres from Japan's Dome Fuji Station, which the SCAR Composite Gazetteer of Antarctica places at 77.32 degrees south, 39.70 degrees east and 3,810 metres. (The gazetteer itself stores that point as GeoJSON, longitude first.) The hotel dropped out of the East Coast accumulation report, the total of

insured values the insurer holds in one region, until someone looked at the map and asked why a hotel was standing on the ice. A model asked "which insured locations are within 5 kilometres of the fire?" would have answered from the same map layer and missed it.

The loss is MEANING: the numbers arrived intact and the rule that said which one was latitude did not. A Type Contract with the CRS identifier on both sides, and a test that the point lies in the area its address names, holds the point at the map build instead.

ERA CARD

Reader. From the clerk reading an address to a map script and a model reading a pair of numbers; each later reader has less context to notice that a hotel cannot be in Antarctica.

Security. Location is often personal data, and coordinates with six decimals can identify a single house; the contract's precision line is also a privacy control.

Cost. One wrong pin costs nothing to store; the cost is in accumulation, pricing and catastrophe reports that silently leave the risk out.

AI. A model answering a distance or "near" question computes with whatever order the table had; it needs the CRS identifier with the values, or it inherits the swap.

Monday rule: for every table and file that holds coordinates, record the CRS identifier beside the columns and add one test that the point lies inside the region its address names.

CHAPTER TEN

Absence



In December 1986 E. F. Codd opened a paper in *SIGMOD Record* with a complaint about his own field: "There has been some technical and justified criticism of the treatment of missing information in the data sublanguage SQL and in IBM's Database 2 system", some of it "directed (by mistake) at the relational model". Thirty-nine years later, in the insurer's claim file for the hotel fire, a field called "last smoke detector inspection" was empty, and the claims-triage agent wrote that the detectors had never been inspected. They had been, in August 2025. The empty field made no claim at all; the reader made one for it. The decision this chapter changes: **every field that can be empty says, in its contract, which kind of empty it holds, and a reader that is not told stops instead of choosing**.

Nine kinds of empty

"Missing" sounds like one thing. In a record it is at least nine different claims, and each one supports a different action:

- **Field absent**: the source did not send the field at all. Nothing is said.
- **Null**: the field is there and carries a mark instead of a value. What the mark means is up to the system.
- **Empty string**: a value of length zero. In a name field it can be a real answer.
- **Zero**: a measured or counted nothing. Zero claims made last year is a fact.
- **NaN**: "not a number", the IEEE 754 result of an invalid operation, such as infinity minus infinity or zero divided by zero.
- **Unknown**: the property applies and has a value, but nobody has it yet.
- **Not applicable**: the property has no proper value for this thing: the inspection date of a building with no detectors.
- **Not done**: the act the field records did not happen. "Not inspected" is a finding, not a gap.

- **No record found**: this source has nothing; another source may.

These are different claims, not different spellings of one claim. "Not done" can void a warranty; "unknown" sends someone to ask; "no record found" sends someone to the next system. Figure 10.1 sets them against the formats of this book and against the newest reader. The pattern is plain: the formats give native forms to most of the first five, which describe the field, and almost none to the last four, which describe the world.

FIGURE 10.1

Kinds of empty across SQL, CSV, JSON, Parquet and the model: nine claims, few native forms

Claim	SQL	CSV (RFC 4180)	JSON (RFC 8259)	Parquet	The model reading it
Field absent: the source never sent it	No in-row form: every row has every column	Short row or missing column; the RFC says lines should have the same number of fields	Member omitted; meaning set by the application (in an RFC 7396 merge patch, leave unchanged)	Column not in the file's schema	Sees nothing; cannot tell that a field was expected
Null: a mark, no value	NULL; comparisons with it yield unknown	No form; any convention is private	The literal null; meaning not defined by the RFC	Optional field; nullity in definition levels, no value stored	Reads a word such as null or None and supplies a meaning
Empty string: a value of length zero	'' (Oracle Database treats a zero-length character value as null)	Empty field: the same bytes many writers use for null	""	Zero-length BYTE_ARRAY value	Easily read as missing, or as nothing to report
Zero: a measured or counted nothing	0; SUM of no rows is NULL, not 0	0	0	0, with the column's type	A fact; may be confused with absent if the export wrote 0 for empty
NaN: a numeric result with no value	Vendor-defined (PostgreSQL sorts NaN as equal to NaN)	Text such as NaN; some readers turn NA and empty into NaN	Not permitted in the grammar	A FLOAT or DOUBLE value; nan_count statistic	Reads the letters, not the IEEE 754 rule

Claim	SQL	CSV (RFC 4180)	JSON (RFC 8259)	Parquet	The model reading it
Unknown: applies, not yet known (Codd's A-mark)	NULL (the same mark)	Empty field	null or a reason code	Null	Cannot tell unknown from the other empties
Not applicable: no proper value exists (I-mark)	NULL (the same mark)	Empty field	null, omitted or a reason code	Null	May treat it as a gap to fill
Not done: the act did not happen	Needs a status column	Needs a status column	Needs a status member	Needs a status column	May read any empty as not done
No record found: nothing in this source	Zero rows: false or unknown depends on the world assumed	No line	Empty array, or a not-found response	No row	May read silence as never happened

Only a few of the nine claims have a native form in any format; the rest need a status field or a reason code, or they collapse into one empty that every reader interprets for itself.

Author's table based on PostgreSQL 19 and Oracle Database (release 26) documentation, RFC 4180, RFC 8259, RFC 7396, the Apache Parquet format specification (README and parquet.thrift), ECMA-262 and HL7 FHIR R4 (data-absent-reason); the model column states risks, not measured rates.

Codd's marks: unknown and inapplicable

Codd met the problem in the relational model and wrote about it three times. In "Extending the database relational model to capture more meaning" (*ACM Transactions on Database Systems* 4(4), December 1979), the extended model called RM/T, section 2.3 gives nulls a three-valued logic: a comparison with a null is neither true nor false but takes a third truth value, and each join and division gets a TRUE version and a MAYBE version. Nulls that operators generate are "always of the type 'value at present unknown,'" and the paper names the alternative: under a closed-world reading "the 'property inapplicable' type would be more appropriate". In section 3 it makes the point that matters for "no record found": a

missing row can be read as "(1) unknown; (2) false". The first is the open world interpretation, the second the closed world one. A query that finds no inspection row says "never inspected" only if the system is declared closed.

The 1986 paper, "Missing information (applicable and inapplicable) in relational databases" (*SIGMOD Record* 15(4), pages 53 to 78), split the null in two: "A missing and applicable" and "I missing and inapplicable". The **A-mark** says a value exists and is not yet known; the **I-mark** says the property does not apply. Codd insisted on a distinction this chapter relies on: "The semantics of the fact that a db-value is missing are NOT the same as the semantics of the db-value itself." He argued against the alternative of picking a special value per column to mean missing, because it "treats each missing db-value as if it were just another db-value" and leaves the handling "buried in the application programs". The paper notes that its material was planned for a book; *The Relational Model for Database Management: Version 2* (Addison-Wesley, 1990) gives missing information two chapters, 8 and 9.

SQL as products implement it kept one mark, NULL. That is the root of most absence defects in databases: a single NULL carries unknown, not applicable, not done and "the export never filled this", and the reader cannot tell which.

Three-valued logic in daily SQL

The SQL standard is ISO/IEC 9075; its current Part 2, SQL/Foundation, is the 2023 edition. Vendor documentation states the consequences most clearly. PostgreSQL's: "SQL uses a three-valued logic system with true, false, and null, which represents 'unknown'", and "Ordinary comparison operators yield null (signifying 'unknown'), not true or false, when either input is null. For example, 7 = NULL yields null." Oracle's adds the rule that surprises people: NOT FALSE is TRUE, but "NOT UNKNOWN evaluates to UNKNOWN". A WHERE clause keeps only rows where the condition is true, so a row with a null falls out of a query and out of its negation:

```
-- SELECT hotel FROM inspections, with each WHERE:
-- inspected is NULL for the hotel
WHERE inspected < DATE '2025-01-01'        -- not returned
WHERE NOT (inspected < DATE '2025-01-01')  -- not returned
WHERE inspected IS NULL                    -- returned
```

Aggregates follow the same rule. In PostgreSQL `count(*)` counts rows, while `count(column)` counts rows "in which the input value is not null"; `avg` averages only non-null inputs; and "sum of no rows returns null, not zero as one might expect". A dashboard that shows "inspected hotels: 58 of 60" may be counting nulls out, not inspections in.

Products also differ on the edges. Oracle Database "currently treats a character value with a length of zero as null", and Oracle itself recommends not relying on it; a column copied into Oracle from another database loses the difference between an empty string and a null. Oracle's documentation also says what every contract should: "do not use null to represent a numeric value of zero, because they are not equivalent." PostgreSQL, so that values can be sorted and indexed, "treats NaN values as equal, and greater than all non-NaN values", while IEEE 754 "specifies that NaN should not compare equal to any other floating-point value (including NaN)". The same comparison gives different answers in a database and in the program that reads from it; the ECMAScript specification, for one, defines Number equality to "return false" whenever either side is NaN.

Empty in the formats

CSV. RFC 4180, the informational description of CSV, has no null at all. A field is text between commas; an empty field is zero characters; "Spaces are considered part of a field". Every meaning of an empty CSV field is a private agreement. Readers fill the gap with their own defaults. With default settings the pandas library reads the empty string, "NA", "NULL", "N/A", "None" and several other spellings as NaN; computed for this book with pandas 2.2.3, a country column holding NA, the ISO 3166-1 two-letter code for Namibia, comes back as NaN. The empty inspection date and Namibia became the same value.

JSON. RFC 8259 gives JSON a `null` literal and does not say what it means. It also allows a member to be left out. So JSON has two native forms of empty, and their meaning is set by each application. One standard does set it. JSON Merge Patch (RFC 7396) says "Null values in the merge patch are given special meaning to indicate the removal of existing values in the target", while a member the patch omits stays as it was. In that protocol an omitted member means "leave it" and a null means "delete it", the opposite of what many teams assume. JSON's grammar does not permit NaN at all, so a serializer that meets one can fail, write null or write a string, and the reader is not told which.

Parquet. Parquet makes absence part of the schema. A field is REQUIRED, which "can not be null", or OPTIONAL, which "can be null". For optional fields, "Nullity is encoded in the definition levels ... NULL values are not encoded in the data": the file records whether each value is present, not why it is missing. The metadata applies the same discipline to itself. The specification says readers "MUST distinguish between null_count not being present and null_count == 0", and if the NaN count is not present, "readers MUST assume NaNs may be present". That is the rule this chapter asks of every field: an absent count is not a zero count.

A reason code. HL7 FHIR carries the reason as data. It defines a data-absent-reason code system "used to specify why the normally expected content of the data element is missing", with codes that include unknown, asked-unknown, not-applicable, not-performed, masked

and error, and even not-a-number. Its not-performed is exact: "The value is not available because the observation procedure (test, etc.) was not performed." It is the A-mark and the I-mark with a vocabulary, and it shows the fix for every other format: when the kind of empty matters, send it as a value of its own.

How a model reads an empty field

A language model receives the field as text: `last_inspection:` , or `"last_inspection": null` , or nothing at all. None of these carries the kind of empty. A database at least returns unknown and stops the comparison; a model asked "were the detectors inspected?" is asked to produce an answer, and the surrounding words make some answers more plausible than others. In a claim file about a fire, an empty inspection field sits next to words that make "never inspected" a fluent reading. This is a risk, not a measured rate: how often a given model turns an empty into a finding depends on the model, the prompt and the context. The contract cannot depend on the model guessing well, and it does not have to. It can send the reason code with the value, and it can require the answer this chapter keeps returning to: "not recorded in this source", with the name of the source that should have it.

> **OUR VIEW, AS OF NOVEMBER 2026**
>
> Absence is the line of the Type Contract most often left blank, and the one a model is most likely to fill. A field that cannot say why it is empty should not reach an agent that writes findings.

TYPE CONTRACT: ABSENCE (THE INSPECTION DATE OF HOP 10)

1. Type, precision and rounding. A calendar date, the day the last smoke detector inspection was completed, plus a status value. *Test:* a value that is not a valid date, or a date in the future, is held.

2. Encoding and serialization. ISO 8601 date in text; in JSON, the member is always present, with the date or null plus the status; in CSV, an explicit status column, never a bare empty field. *Test:* an export row with an empty date and no status fails the export.

3. Unit or currency. Not applicable. *Test:* none needed.

4. Time basis. Local date at the hotel, zone America/New_York; the status is as at the export time, which travels with the file. *Test:* a file without an export timestamp is held.

5. Identity scope. The detector system of the insured building, the hotel of policy CP 0041-7730. *Test:* the record resolves to that building, not to the hotel group.

6. Absence. The heart of this contract. Every record carries one status: `done` (date required) or, with a null date, `not-recorded-here` (the inspection may exist in the inspection vendor's system, which is named), `not-done`, or `not-applicable` (no detectors in scope). The empty string is not a value. No record found in this system means "not recorded here", never "not done". *Test:* a null date without a status is held; a reader, human or model, that turns `not-recorded-here` into "never inspected" fails the boundary test, and the agent's test prompt must return "not recorded in the facilities system; check the inspection vendor's record".

7. Authority. The hotel facilities manager owns the field; the inspection vendor's own system is the record of the inspection and wins on conflict. *Test:* a status of `not-done` requires a named person to set it.

THE HOP · 10 · 2025 · FACILITIES SYSTEM TO CLAIM FILE

Record	Last smoke detector inspection (One Night, Every Value, illustrative), hop 10, 2025
From	Facilities system
To	Claim file
Mechanism and schedule	The inspection date. The export writes an empty string when the field has no value
Format change; what was lost or assumed	COMPLETENESS Empty: the export does not say whether the date was never recorded, the inspection was not done, or the field does not apply. **The inspection was done in August 2025 and recorded in the vendor's own system, not in this field; the agent reads the empty field as 'never inspected'**
Accountable owner	Hotel facilities manager
Who can change it	Facilities staff; the inspection vendor (in its own system)
Classification; retention or erasure duty	Operational safety record
Read by (people, apps, models, agents)	The adjuster, the triage agent, the fire investigator
Cost per month; basis	One-off: a coverage question raised and withdrawn (not monthly) (illustrative)

Hop 10 is that field. The facilities system's export wrote an empty string whenever the inspection date had no value, and the claim file imported it as empty. The export said nothing about why. The date had never been entered in the facilities system, because the inspection vendor keeps its own records: the inspection was done in August 2025 and recorded there. When the triage agent summarised the claim file, it read the empty field as "never inspected" and wrote it into its summary. An adjuster raised a coverage question, then withdrew it when the vendor's record was found.

The loss is COMPLETENESS: the claim file received neither the inspection date nor a reason for its absence. The date lived in the vendor's system, not in the field the export read, and the empty that arrived stood for "not recorded here", one of three meanings (not recorded here, not done, not applicable) the format could not tell apart. Each step was correct by its own

rules. The export wrote what it had; the claim file stored what it got; the agent produced a fluent reading. With line 6 of the contract the export writes `not-recorded-here` and names the vendor, and the agent's answer becomes a question to the right system.

ERA CARD

Reader. SQL gave absence one mark and three truth values; the web formats gave it two spellings; the model reads it as a gap in a sentence.

Security. Absence hides failure: a control that was never run and a control that passed can look the same in a report, and an empty field is an easy place to hide a skipped step.

Cost. A status column costs a few bytes per row; a finding written from an empty field costs a review, a coverage dispute or a wrong decision about a person.

AI. A model must be told which kind of empty it is reading, and told to answer "not recorded in this source" rather than infer; give it the reason code, not the blank.

Monday rule: pick the ten nullable fields your agents read most, add a status value that says which kind of empty each null is, and make every export fail on an empty field without one.

CHAPTER ELEVEN

Records as Text: Fixed Width, CSV, XML and JSON



At 05:42:10 UTC on 2 November 2025 fire panel 2 wrote an alarm into its event log. Days later the alarm reached the insurer's claim file as one line of a CSV export, and the time arrived as nineteen characters: `2025-11-02 01:42:10`. Nothing in the file said that the digits were a wall-clock reading, which zone the clock was in, or which of the two 01:42s of that night it meant. The file was not broken. It was a text format doing what text formats do: carrying characters and leaving the rules somewhere else. Part II looked at kinds of value. Part III looks at the containers that carry them, and starts with the four text formats every enterprise still exchanges every day. The decision this chapter changes: **choose a text format by what it can say about a value, and write down, next to the file, every rule it cannot say**.

Six questions a format can answer, or not

A text format turns a record into characters. A reader turns the characters back into values. Between the two, the reader needs answers to the questions the Type Contract asks. Six of them decide whether a text file can carry a value safely: what **type** each field is (number, date, text); its **scale**, the number of decimal places that matter; its **unit** or currency; its **time basis** (an instant, a wall time or a date, and in which zone); how **null**, the absence of a value, is written; and the character **encoding** of the bytes.

A format answers a question when the answer is in the file, in a form a program can read without a person's help. It leaves the question open when the answer sits in a layout document, an email, a header word or a habit. This chapter asks the six questions of fixed-width files, CSV, XML and JSON, and Figure 11.1, at the end, puts the answers side by side. How people use these formats on the desk, and above all what the spreadsheet does to a CSV file, is covered in depth later in this series. Here the question is narrower: what does the file itself know?

Fixed width: the layout lives outside the file

The oldest record format still in daily use gives each field a fixed number of character positions. A record is a line; field three is whatever sits in positions 18 and 19. Fixed width came from the punched card and the tape, and the copybook that described it is told in *From Tally to Token*. It survives in interfaces between companies and in the nightly exports of building systems such as the hotel's fire panel vendor. Here is the alarm as a fixed-width record, in a layout invented for this example:

```
ALM20251102014210 02

Layout (a separate document):
pos  1 to 3   event code      ALM = alarm
pos  4 to 17  event time      YYYYMMDDHHMMSS, panel clock
pos 18 to 19  panel number    two digits, zero filled
```

The record holds nineteen characters and not one rule. That the first three are a code, that the next fourteen are a time, that the time is the panel's local clock, and that `02` is a panel number and not a quantity: all of it lives in the layout. A decimal amount in such a file usually has no decimal point at all; the layout says "last two positions are cents", and the file shows `0123456`. Absence is a convention too: spaces, zeros, or a field of nines, chosen by whoever wrote the first program. The encoding is agreed by the two parties, often in a contract signed years before anyone now working there arrived.

That is the defining property of the format, not a defect of one file. A fixed-width record cannot be read at all without its layout, so the layout tends to be kept, versioned and sent along. Its weakness is the opposite of what people fear: it rarely fails loudly. Read with last year's layout, a record still parses into fields, and every field still holds characters of the right length.

> **WHAT HASN'T CHANGED**
>
> From the card layout sheet to the CSV header row to the API document, the rules of a text record have mostly lived outside the record. Each new format moved a few of them inside. None moved all six.

CSV: commas, quotes and nothing else

Comma-separated values, in the words of Python's documentation, were "used for many years prior to attempts to describe the format in a standardized way in RFC 4180". The description most people cite is RFC 4180, "Common Format and MIME Type for Comma-Separated Values (CSV) Files", published in October 2005 in the IETF's **Informational**

category: it documents common practice and registers the media type `text/csv`; it is not an Internet Standard. Its rules fit on one page. Each record is on its own line, "delimited by a line break (CRLF)". A first line may be a header. Fields are separated by commas. A field that contains a comma, a double quote or a line break is enclosed in double quotes, and a double quote inside such a field is written twice.

That is all. RFC 4180 defines no data types: every field is text. It defines no scale, no unit and no time basis. It has no null: an empty field is a field of zero characters, and the format cannot tell "no value" from "the empty string" from "not exported". The encoding is not in the file. The media type registration allows an optional `charset` parameter and notes that "common usage of CSV is US-ASCII"; but the parameter travels in an email or an HTTP header, not in the bytes, and a file saved to a share has lost it. Even the header is optional, signalled by a `header` parameter with the values "present" or "absent" that most files never carry.

The night's CSV export shows the consequence in one line:

```
panel,event,time
2,ALM,2025-11-02 01:42:10
```

A reader cannot tell from the file whether `2` is a number or a label, or what `time` means. Python's standard library is honest about it: its csv module returns "each row … as a list of strings", and "no automatic data type conversion is performed" unless the caller picks a quoting option that converts: one turns every unquoted field into a binary float, and two added in Python 3.12, which the reader honours from Python 3.13, turn an empty unquoted field into None. The same documentation adds that "the lack of a well-defined standard means that subtle differences often exist in the data produced and consumed by different applications". Every type a CSV value has at the far end was given to it by the reader. The guest's name took the same road: the property system's export converted it to Windows-1252, which has no ı or ş, so those letters were replaced, and nothing in the file said so (Chapter Five).

On each platform. RFC 4180 says CRLF; tools on Unix-like systems often write a bare line feed, and both kinds of file arrive in the same landing zone. Python's documentation warns that if a CSV file is not opened with `newline=''`, "newlines embedded inside quoted fields will not be interpreted correctly, and on platforms that use \r\n line endings on write an extra \r will be added". A line ending is a value rule too: a quoted address with a line break inside it is one field to one reader and two broken records to another.

XML and XML Schema: types, if you bring the schema

The Extensible Markup Language, XML 1.0, became a W3C Recommendation on 10 February 1998. It answered one of the six questions inside the file for the first time: the encoding. Every XML processor must read UTF-8 and UTF-16, and a document in any other encoding must say so in its declaration, as in `<?xml version="1.0" encoding="ISO-8859-1"?>`. It also settled line endings for its readers: section 2.11 requires a processor to pass every line break to the application as the single character line feed. Names travel with values, so `<alarmTime>` says more than column three of a CSV file.

But a name is not a type. XML 1.0 defines no data types for element content; its document type definitions can type attributes only, as strings, identifiers, tokens or enumerations. The types arrived with **XML Schema Part 2: Datatypes**, a W3C Recommendation of 2 May 2001. It defines `decimal`, which "represents arbitrary precision decimal numbers", with facets for the total number of digits and the digits after the point; the same edition requires processors to support at least 18 decimal digits. A schema can say that an amount is a decimal with at most two fraction digits, and a validating reader will reject 1234.567. The facet limits the value, not the written form: 1234.5 and 1234.50 are the same decimal, so the scale as written is not kept. For absence, the companion Part 1 offers `xsi:nil="true"`: an element declared nillable "may be valid without content". An empty element and a nil element are different statements, if the schema says so.

Time is where XML Schema is most instructive. Its `dateTime` "represents a specific instant of time", yet the timezone in its lexical form is optional. The alarm can be written as a valid dateTime with no offset:

```
<alarmTime>2025-11-02T01:42:10</alarmTime>
```

The specification knows this is not an instant. It gives values without a timezone only a partial order: comparing `2000-01-16T12:00:00` with `2000-01-16T12:00:00Z` is, in its own example, indeterminate, because the first could lie anywhere within 14 hours either side. A reader that compares the alarm with an arrival time is told by the standard that it cannot know the answer, and many programs simply compare the digits. XSD 1.1 (W3C Recommendation, 5 April 2012) added `dateTimeStamp`, whose values "are required to have explicit time zone offsets", so that "the datatype is totally ordered". A schema that uses it would have refused the panel's text at the boundary. An offset is still not a zone: neither type can say `America/New_York`.

Three cautions keep XML Schema in proportion. The types live in a separate schema document, so a file and its schema can part company like a fixed-width file and its layout. Validation is a choice of the reader. And no datatype carries a unit or a currency; those travel as element names or attributes whose meaning, again, is agreed outside the file.

JSON: numbers without a stated precision

JSON is defined twice, with one grammar: RFC 8259 (December 2017, Internet Standard STD 90) and ECMA-404, second edition, December 2017. ECMA-404 states its aim plainly: "only to define the syntax of valid JSON texts", without semantics. JSON has, in RFC 8259's words, "four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays)". It answers some of the six questions. A number is distinguished from a string, so `2` and `"2"` differ. Null is a literal, and a missing member is a second, different kind of absence. The encoding is fixed: RFC 8259 section 8.1 says JSON exchanged between systems outside a closed ecosystem "MUST be encoded using UTF-8", and implementations "MUST NOT add a byte order mark" to it.

What a number means is left to the reader. RFC 8259 section 6 "allows implementations to set limits on the range and precision of numbers accepted", observes that IEEE 754 binary64 is widely available, and warns that a number such as 1E400 or 3.14159265358979323846264338327 "may indicate potential interoperability problems". The RFC calls integers between minus (2^53 minus 1) and 2^53 minus 1 interoperable: implementations will agree exactly on their values. The grammar carries no scale: `1234.56` and `1234.560` are both valid, and nothing says which digits matter. The refund of the night, EUR 1,234.56, went to the payment API as a JSON number, and downstream it was summed in binary64, in which the nearest value to 1234.56 is 1234.55999999999994543031789362430572509765625 (Chapter Four tells the cents).

The same text can give two readers two values. We ran one line through two standard parsers on 20 November 2026:

```
{"id": 9007199254740993}

Node.js 22.20.0  JSON.parse   -> 9007199254740992
Python 3.13.3    json.loads   -> 9007199254740993
```

JavaScript reads every JSON number into its Number type, a binary64, and the last digit changes without an error. Python keeps integers exact but reads `1234.560` as the float 1234.56 unless told to use its decimal type. Both readers conform. The **I-JSON** profile, RFC 7493 (March 2015), turns that experience into rules for interoperable messages: they "SHOULD NOT include numbers that express greater magnitude or precision than an IEEE 754 double precision number provides", larger exact values are "RECOMMENDED" as strings, and dates and times are recommended as RFC 3339 strings with an explicit timezone and seconds. An amount sent as `{"amount": "1234.56", "currency": "EUR"}` keeps its digits and its currency; the type is then a string, and the rule that it is a decimal with two places must be written somewhere else.

Time has no JSON type at all. The alarm travels as a string, and `"2025-11-02 01:42:10"` is as valid a string as `"2025-11-02T05:42:10Z"`. JSON Schema, the subject of the next chapter, can say `"format": "date-time"`, which means RFC 3339 with a required offset; but in its 2020-12 version format checking as an assertion "MUST be disabled by default", so a schema-valid file can still carry the panel's text unless the reader turns the check on.

Duplicate names: when two parsers read two records

One more rule of RFC 8259 is a security boundary. Section 4 says that "the names within an object SHOULD be unique", and then describes what happens when they are not: "the behavior of software that receives such an object is unpredictable. Many implementations report the last name/value pair only. Other implementations report an error or fail to parse the object, and some implementations report all of the name/value pairs, including duplicates." A SHOULD is not a MUST, so `{"role": "user", "role": "admin"}` is valid JSON. When two components of one system parse the same text differently, an attacker can show each component a different record. This is called a **parser differential**.

> **SOURCED FAILURE: CVE-2017-12635, APACHE COUCHDB**
>
> The CVE record, published on 14 November 2017, describes the case in one paragraph. "Due to differences in the Erlang-based JSON parser and JavaScript-based JSON parser", Apache CouchDB before 1.7.0 and 2.x before 2.1.1 accepted user documents with two `roles` keys. "The second one will be used for authorising the document write, but the first 'roles' key is used for subsequent authorization for the newly created user." By design users cannot assign themselves roles; with the differential, "the vulnerability allows non-admin users to give themselves admin privileges". The JSON was valid. Two readers took two values from it.

I-JSON closes the gap for the messages that adopt it: objects "MUST NOT have members with duplicate names". A boundary that accepts JSON from outside should reject duplicates before any component reads the record, and should parse once, passing the parsed result on rather than the text. The same discipline applies to numbers: if a gateway and a ledger read `9007199254740993` differently, they are two readers of two amounts.

Figure 11.1 sums up what each format carries. Read the columns, not the rows: no text format, with or without a schema, has a built-in place for a unit or a time zone name.

FIGURE 11.1

What each text format knows about a value: a schema adds type, null and limits on digits; none has a built-in place for unit or zone

Format	Type	Scale	Unit or currency	Time basis	Null	Encoding
Fixed width	Only in the layout document	Implied by the layout ("last two positions are cents")	Layout or nowhere	Layout or nowhere	A convention: spaces, zeros or nines	Agreed between the parties; not in the file
CSV (RFC 4180)	None: every field is text	None: digits as written	None; a header word at best	None	None: an empty field is zero characters	Not in the file; optional charset parameter on the media type only
XML with XML Schema	Yes: decimal, dateTime, date, boolean and more	Limits only: totalDigits and fractionDigits; trailing zeros are not kept	None in the datatypes; element names or attributes by agreement	dateTime offset optional; dateTimeStamp (XSD 1.1) requires one; never a zone name	xsi:nil on nillable elements; absent element	In the file: UTF-8 or UTF-16, or declared
JSON (RFC 8259)	String, number, boolean, null, object, array; no date type	None; precision left to the reader (binary64 is common)	None	None: times are strings	null literal, plus the absent member	UTF-8 required between open systems; no BOM
JSON with JSON Schema 2020-12	Yes, including integer	Only as a check (multipleOf); no declared scale	None	format date-time requires an offset, but format is not asserted by default	type null allowed or not; required lists members	UTF-8, from JSON

Each cell says whether the answer travels inside the file or its schema; where it does not, the reader must get it from somewhere else.

Source: RFC 4180 (October 2005); XML 1.0 (W3C, 10 February 1998); XML Schema Part 1 and Part 2 (W3C, 2001; second edition 2004) and XSD 1.1 Part 2 (2012); RFC 8259 and ECMA-404 (December 2017); RFC 7493; JSON Schema Validation 2020-12.

A schema moves type, limits on digits and null from the reader's habits into a document the reader can check, which is why the next chapter is about schemas that travel. Unit and time basis stay outside every one of these formats. A Type Contract for a text field therefore always has at least two lines the file cannot hold, and the boundary test has to read them from somewhere else: a contract stored with the feed, a schema annotation, or a field of its own such as `currency` or `tz`.

OUR VIEW, AS OF NOVEMBER 2026

CSV and JSON will stay the most common ways companies hand each other records, because they are easy to write. The safe habit is to treat both as transport, never as a description: a CSV file without its contract, or a JSON number without its scale and currency, is a value half delivered.

ERA CARD

Reader. Every era's reader of exchanged records: the tape job with a layout, the loader with a header row, the validating XML parser, the web service, and now the agent that reads a CSV export as plain text.

Security. Parsing is attack surface. Duplicate names and number limits let two components read two values from one text (CVE-2017-12635); parse once at the boundary, reject duplicates, and pass on the parsed record.

Cost. Text formats are nearly free to produce and expensive to read correctly: every reader rebuilds the rules the file left out, and each rebuild is a chance to differ.

AI. A model reading a CSV line or a JSON string sees the same nineteen characters a parser sees and no more; a missing offset or currency is not in the text for it to find, so the contract must be sent with the record.

Monday rule: for every CSV or JSON feed that crosses a company boundary, store next to it a contract that names the encoding, the type and scale of each number, the unit or currency and the time basis of each time, and reject JSON with duplicate member names at the gateway.

CHAPTER TWELVE

Schemas That Travel



THE ERAS OF THE RECORD

The refund of the night left group finance as a message to the payment service carrying an amount, 1234.56. Suppose the payments team had replaced its JSON with the best tools of this chapter. The message is a binary record with a schema; the schema is registered; the registry checks every change against the last version; and the amount is a decimal with two places, not a float. The message is still missing one fact the reader needs: that the amount is in euros. None of the general-purpose schema languages in this chapter attaches a currency to its decimal type. A currency field can be added beside the amount, but only by design, and no compatibility check would notice if finance changed the euros to dollars tomorrow. This chapter looks at the schema languages that do travel with data, what they settle, and what they leave open. The decision it changes: **use a schema with a registry for every stream that crosses a team boundary, and put the lines a schema cannot check into the contract beside it**.

Two schemas, not one

A **schema** is a machine-readable description of a record: its fields, their types and, sometimes, their defaults. Chapter Eleven showed what a schema adds to a text format. Binary formats go further: their bytes cannot be turned back into named, typed fields without a schema, so the schema has to reach the reader by some route. There are three routes. The schema can be compiled into the reading program, written into the file, or stored in a registry and referenced by a number in each message.

Every route meets the same fact of enterprise life: the writer and the reader change at different times. A record written last year under one version of the schema is read today by a program built for the next version. The useful question is therefore not "does the data match the schema?" but "what does this reader do with data written under that schema?" Each of the languages in this chapter answers it differently, and the answers decide which changes a company can make without stopping every consumer at once. Figure 12.1, later in the chapter, sets the answers side by side.

ASN.1: the notation and its encodings

The oldest schema language still in wide use is **Abstract Syntax Notation One**, ASN.1. Recommendation X.208, "Specification of Abstract Syntax Notation One (ASN.1)", was issued in November 1988 by the CCITT, the predecessor of ITU-T, with its companion X.209 for the Basic Encoding Rules; both were withdrawn on 30 October 2002, superseded by X.680 to X.683 and X.690. The current notation is X.680, edition of February 2021. Among much else, its notation specifies the structure of the X.509 certificates that secure web connections present.

Its central idea is the one this book is about. ASN.1 separates the **abstract syntax**, what the values are, from the **transfer syntax**, how they become bytes, and it publishes several sets of encoding rules for one notation. X.690 defines three. The Basic Encoding Rules (BER) give the sender "various choices as to how data values may be encoded"; clause 7.3 says "alternative encodings are permitted by the basic encoding rules as a sender's option". The Canonical and Distinguished Encoding Rules (CER and DER) "select just one encoding from those allowed by the basic encoding rules". That matters when a value is signed. A signature covers bytes, not meaning, so two encodings of one certificate would carry two different signatures. RFC 5280, the internet profile for X.509, therefore requires that the data to be signed "is encoded using the ASN.1 distinguished encoding rules (DER)".

X.691 defines the Packed Encoding Rules (PER), which produce compact encodings by leaving out what the schema already knows. Its clause 7.2 states the price: "Without knowledge of the type of the value encoded, it is not possible to determine the structure of the encoding." A PER message without its schema is not a damaged record; it is noise.

ASN.1 also shows where a schema stops. Its UTCTime type carries a two-digit year. The type does not say which century a year belongs to; RFC 5280 does, for certificates: a year of 50 or more is read as 19YY, less than 50 as 20YY, and validity dates from 2050 on must use GeneralizedTime. The rule that makes the value usable lives in a profile, a separate document written for one use of the type. Profiles are how industries add Type Contract lines to a schema language that lacks them; Chapter Thirteen meets them again in health and payments.

Protocol Buffers: numbers on the wire, names in the code

Protocol Buffers, usually shortened to protobuf, is in its documentation's words "a language-neutral, platform-neutral extensible mechanism for serializing structured data". A field is declared with a name, a type and a **field number**, and the encoding guide is explicit that only the number travels: the binary form "just uses the field's number as the key". Each field on the wire starts with a tag that combines the field number with one of six **wire types** (VARINT, I64, LEN, SGROUP, EGROUP and I32) that tell a parser how to find the length of

the value, so it can skip a field it does not know. Names and declared types are known only to a reader holding the `.proto` file. The proto3 guide allows field numbers from 1 to 536,870,911, reserves 19,000 to 19,999 for the implementation, and warns that a number "cannot be changed once your message type is in use".

That design makes some changes safe and others silent. Because each field carries its wire type, "old parsers … skip over new fields they don't understand", and proto3 messages preserve those unknown fields when they write the message on. The guide calls adding and removing fields safe on the wire; a removed number must be reserved so that no one reuses it. Renaming a field is safe in binary, where the name never travels, but not in the JSON and text formats, where it does. Integer widths are "compatible", with a caveat in the guide's own words: "if a 64-bit number is read as an int32, it will be truncated to 32 bits". The parse succeeds and the value is different. One more rule echoes the last chapter's parser differentials: if a field appears twice in one message, "the parser accepts the last value it sees".

Absence is the subtle part. In proto3 a field with **implicit presence** is not written when it holds its default (zero, the empty string, false), so, as the guide puts it, "you cannot determine whether the default (zero) value was set or parsed from the wire or not provided at all". A detector inspection count of zero and an inspection count nobody recorded arrive as the same nothing, the same ambiguity Chapter Ten found in the night's empty inspection date. Explicit presence came back to proto3 with the `optional` keyword, behind an experimental flag from release 3.12.0 and on by default since 3.15.0; the guide now recommends it. For JSON, protobuf's own mapping writes 64-bit integers as decimal strings and timestamps as RFC 3339 text "Z-normalized", which avoids the binary64 problem Chapter Eleven showed and also turns every time into UTC on output.

Avro: the writer's schema meets the reader's

Apache Avro took the opposite route. Its binary encoding "does not include field names" or field numbers: values follow one another in schema order, and nothing in the bytes says where one ends without the schema. So the schema must travel. An Avro container file stores "the schema of objects stored in the file" in its header, under the key `avro.schema`; a single message can carry the two bytes `C3 01` and an eight-byte fingerprint of its schema instead.

What Avro adds is a written procedure for change. The specification (version 1.12.0) calls the schema used to write the data the **writer's schema** and the one the application expects the **reader's schema**, and it defines **schema resolution** between them. Fields are matched by name. A field the writer has and the reader lacks is ignored. A field the reader has and the writer lacks takes the reader's default; without a default, resolution fails with an error. An int

may be read as a long, float or double, but not the reverse. An enum symbol the reader does not know becomes the reader's default symbol, or an error. A renamed field resolves only if the reader lists the old name in its `aliases`.

Avro's **logical types** carry some of the Type Contract inside the schema. A `decimal` annotates bytes with a precision and a scale, and stores the unscaled integer, so the refund is the integer 123456 with scale 2. For resolution, two decimal schemas match only "if their scales and precisions match": a reader cannot silently take a scale-4 amount as scale 2. Time is split the way Chapter Six split it. `timestamp-millis` and its micro and nano siblings are "an instant on the global timeline, independent of a particular time zone or calendar"; `local-timestamp-millis` is "a timestamp in a local timezone, regardless of what specific time zone is considered local". The panel's alarm text is the second kind. A schema that declares it the first kind writes a false instant, the same mistake Chapter Fourteen finds in the night's warehouse files.

Two rules in the specification deserve a line in any review. A logical type that is invalid, such as a decimal whose scale exceeds its precision, is to be ignored, and the reader falls back to the underlying type: the amount arrives as raw bytes, without an error. And a field's `doc` text, the natural home for a unit or a meaning, is "ignored for the purposes of schema resolution". Whatever a team writes there, no reader is required to read it.

JSON Schema and the registries

JSON Schema describes JSON documents. Its current version is 2020-12. It can require a member, constrain a number's range, and declare a string's `format`, including `date-time`, the RFC 3339 production with a required offset. Two of its own rules set its limits. On numbers: "The JSON specification allows numbers with arbitrary precision, and JSON Schema does not add any such bounds", so a schema does not stop a reader from turning 1234.56 into a binary64. On formats: checking them as assertions "MUST be disabled by default", so `"format": "date-time"` is a label unless the validator is configured to enforce it. Chapter Fifteen looks at JSON Schema in another role, as the description of tools and outputs given to a language model.

A **schema registry** is a service that stores each version of a schema, gives it an identifier and checks a new version against older ones before accepting it. In Confluent's default wire format, where the schema ID travels in the payload, each message starts with "a version byte, a 4-byte schema ID": the version byte is 0, and the ID, in network byte order, names the registered schema, so the schema travels by reference. The words most teams use for compatibility, **backward, forward and full**, are the registries' terms, not Avro's or protobuf's. Confluent's documentation defines BACKWARD as "consumers using the new schema can read data produced with the last schema", FORWARD as the reverse, FULL as

both, transitive variants that check against every earlier version, and NONE; "the default compatibility mode is BACKWARD". AWS Glue Schema Registry uses eight modes, NONE, DISABLED, BACKWARD, BACKWARD_ALL, FORWARD, FORWARD_ALL, FULL and FULL_ALL, for Avro, JSON Schema and protobuf. Each vendor applies its modes through the format's own rules, which differ: Confluent notes that the rules "vary somewhat based on schema format" and recommends BACKWARD_TRANSITIVE for protobuf.

Read Figure 12.1 down its last two rows. Every check in it compares one schema with another. A change that keeps the schema and alters the meaning gives the registry nothing to compare.

What none of them carries

Put the night's values through the best of these tools and three rules still have no built-in place. The **currency**: the refund is an Avro decimal with scale 2 and a registered schema, and nothing in the schema says EUR. A field named `amount_eur` is a convention; a separate `currency` field with an ISO 4217 code is better, and only a contract says the two must travel together. The **code-system version**: the diagnosis code is a string in every schema language. A record can add system and version fields, as FHIR's Coding does, but no schema language supplies or requires them, so nothing records that the code came from ICD-10-CM, or which fiscal-year edition (Chapter Eight). **What zero means**: a bare integer type, optional or required, cannot say whether zero inspections means none were done or none were counted (a status field beside it can, if someone designs it), and protobuf's implicit presence cannot even keep zero apart from unset.

FIGURE 12.1

Which schema change breaks which reader: the checks see structure, never meaning

Schema change	Avro (schema resolution)	Protocol Buffers	Registry check (vendor terms)
Add a field with a default	New reader fills the default for old data; old reader ignores the new field	Safe on the wire: old parsers skip it; new reader sees the type's default (proto3 has no declared defaults)	Avro: accepted under BACKWARD, FORWARD and FULL
Add a field without a default	New reader fails on old data: resolution error	Proto3 has no required fields: old data reads as zero or empty, indistinguishable from unset	Avro: rejected under BACKWARD; accepted under FORWARD
Remove a field	New reader ignores it in old data; old reader needs its default or fails	Safe on the wire; reserve the number so it is never reused	Avro: BACKWARD accepts; FORWARD only if the field had a default
Rename a field	Matched by name: breaks unless the reader lists the old name in aliases	Binary safe (only the number travels); JSON and text formats break	Run the registry's own check; the result follows the format's rules
Widen int to long	Allowed: int is promotable to long; the reverse is not	Compatible, but a 64-bit value read as int32 is truncated to 32 bits	Follows the format's rules
Change a decimal's scale (2 to 4)	Two decimal schemas match only if scale and precision match	No decimal type: the scale is a convention outside the schema	Follows the format's rules
Add an enum value	Old reader maps the unknown symbol to its default, or fails	Safe on the wire; unknown values are preserved; code that switches on every value may still break	Follows the format's rules

Schema change	Avro (schema resolution)	Protocol Buffers	Registry check (vendor terms)
Reuse a name or number for a new meaning	Old data resolves by name into the new meaning	Forbidden by the guide; old data parses into the new field	Types may match; no check sees meaning
Change the currency or unit, schema unchanged	Nothing to resolve: every reader reads the new unit as the old	Nothing to detect	No new version, so no check runs

Avro and Protocol Buffers each define what a reader does with data written under another version; registry results shown are for Avro under Confluent's modes (other formats follow their own rules), and a change of meaning with no change of schema passes them all.

Source: Apache Avro Specification 1.12.0 (schema resolution, logical types); Protocol Buffers Language Guide (proto 3) and Encoding guide; Confluent Schema Registry and AWS Glue Schema Registry documentation (compatibility modes).

The time basis sits halfway. Avro's logical types and XML Schema's dateTimeStamp can say "instant" or "local", and RFC 3339 can require an offset. None of these schema types can say `America/New_York`, the zone that tells a reader which local hour was repeated on 2 November 2025; a zone field can be added beside the value, and Arrow's in-memory timestamp type (Chapter Fourteen) carries one, but in these schema languages the zone is a design choice, not part of the type. Nor can any of them say who owns a field or who may receive it, the seventh line of the Type Contract.

The practical answer is not a new schema language. It is to use what the schema checks, to name the rest in a contract stored with the schema, and to test the rest at the boundary: currency present and valid, code system and edition present, zone name present, zero distinguished from unknown. Chapter Sixteen shows the tests. The schema settles syntax and structure; the contract settles meaning.

> **OUR VIEW, AS OF NOVEMBER 2026**
>
> Registries made schema change a reviewed act, which is real progress. The next step is to review meaning with the same rigour: a change of currency, unit, code-system edition or time basis should need a new contract version and the same sign-off as a breaking schema change, even when the bytes do not change at all.

ERA CARD

Reader. Programs built at different times from different versions of one schema: the telecom switch with an ASN.1 module, the service with a compiled .proto file, the stream consumer that fetches a schema by its registry number.

Security. Encodings are part of trust. Signatures need one encoding per value (DER); duplicate fields resolve "last value wins" in protobuf; a registry with compatibility NONE, or write access for everyone, lets any producer change what every consumer reads.

Cost. A schema and a registry cost little to run and save the coordinated outage that a breaking change forces; the cost they do not remove is the meaning change that passes every check.

AI. Schemas given to agents carry types and formats, but not currency, code-system edition or what zero means; an agent reading a schema-valid record still has to be given those lines, or it will supply its own.

Monday rule: list every registered schema whose fields hold money, codes, counts or local times, and add to each a contract entry for currency, code-system edition, zone and the meaning of zero, with a boundary test that fails when one is missing.

CHAPTER THIRTEEN

Industry Messages



THE ERAS OF THE RECORD

On 22 November 2025 the coexistence period for cross-border payment instructions on the Swift network ended. From that weekend, in Swift's words, "MT payment instructions are no longer supported by the FIN service for FI-to-FI cross-border flows"; ISO 20022 messages carried them instead. Ten days later, on 2 December 2025, the insurer paid the guest's injury claim to her account in Türkiye, and the payment stopped at its own bank, held for repair. The message was valid against its schema. It failed the payment profile's rules: the town was missing, and the name had been rewritten into the letters the profile allows. An industry message fixes a great deal about meaning, far more than CSV or bare JSON. It does not fix everything, and the rest sits in a usage guideline or an agreement that the sender's system may never have read. The decision this chapter changes: **for every industry message you send or receive, write down which lines of the Type Contract the standard fixes and which the profile or the agreement owns, with its version and its owner.**

What a message standard fixes

An **industry message** is a record format that a whole sector agrees on: its fields, their data types, their code lists and the order of exchange. Chapters Eleven and Twelve looked at general formats and schemas. An industry message goes further, because a committee of the people who use the values has decided what many of them mean.

Every such standard has two layers. The **base standard** defines the structure and the types. A second layer, called a **profile** in health care, a **usage guideline** in payments and an **implementation guide** in insurance and trade, narrows the base for one community: which optional fields become mandatory, which code list edition applies, which characters may appear. Below both sits the bilateral agreement between two parties. Figure 13.1 sets seven messages against that split. The pattern is the same in every row: the standard fixes the type and much of the encoding; the time basis, the code version, the character set and the right to receive are often decided one layer down.

FIGURE 13.1

What each industry message fixes, and what agreements still own

Message	Fixes in the standard	Owned by the profile or agreement	The night
HL7 v2 (hospital interfaces)	Segments and fields; date and time with an optional UTC offset	Whether the offset is sent (only recommended; default is the sender's local zone); code tables and versions; site field usage	Arrival time kept its offset inside the hospital
FHIR R4 and R5	Resources and data types; dateTime with hours and minutes SHALL carry an offset; Coding has slots for system and version	Profiles: which codes, whether the version is filled, which fields are required; who may receive the resource	Offset kept; the loss came later, at the PDF
ISO 20022 pain.001 and pacs.008	Message structure; party, address and amount elements; currency codes	CBPR+ usage guideline: FIN-X text, Latin letters in names and addresses, town and country required, address timetable; bank agreements	Hop 11: town missing, name rewritten into the allowed letters, payment held
ACORD AL3 and XML (P&C)	Policy, commission and claim data elements for members	Implementation guides between carrier, broker and vendor; code list editions in use	Not used: the broker's copy was a spreadsheet
ASC X12 837 (US health claims)	Claim structure; the qualifier ABK names ICD-10-CM for the principal diagnosis	HIPAA version 005010; payer companion guides; the code edition is not in the qualifier	Not used: the liability claim received a PDF bill

Message	Fixes in the standard	Owned by the profile or agreement	The night
XBRL and Inline XBRL	Every number has a unit, a period in its context and a decimals or precision statement	Taxonomy and filing manual of the regulator	Not in the night; the model of a fact that carries its own lines
UN/EDIFACT	Syntax (ISO 9735); segments, data elements and code lists in UNTDID releases	Message implementation guides between trading partners; which directory release	Not in the night

Every message fixes structure and data types; the character set, the mandatory fields, the code versions and who may receive the value are set by a profile, a usage guideline or a contract between the parties.

Source: HL7 Version 2 (DTM data type); HL7 FHIR R4 and R5 Data Types; CBPR+ pacs.008.001.08 usage guideline (SR2025); PMPG Hybrid Postal Address v1.10; ACORD P&C data standards; 45 CFR 162.1102 and CMS diagnosis qualifiers; XBRL 2.1 (sections 4.6 to 4.8); ISO 9735 and UNTDID (UNECE).

The three lanes the night uses get the rest of the chapter: health care, payments and insurance. Financial reporting appears once, as the model of a value that carries its own lines.

Health care: HL7 version 2 and FHIR

Inside the health system the arrival time travelled in **HL7 version 2** from registration, and in FHIR inside the hospital. HL7 version 2 is the family of segment-based messages that hospital systems have exchanged for decades. Its date and time type, DTM, is written as digits from the year down to fractions of a second, followed by an optional offset of the form plus or minus HHMM from UTC. The specification is careful about how optional that offset is: "The HL7 Standard strongly recommends that all systems routinely send the time zone offset but does not require it", and "If the time zone is not included, the time zone defaults to that of the local time zone of the sender." So an HL7 v2 time without an offset is not wrong; it means local time at the sender, and the receiver has to know where the sender is. The health system's registration feed sent the offset. The rule that made the value safe was a local configuration choice, not a requirement of the standard.

FHIR (Fast Healthcare Interoperability Resources) is HL7's resource-based standard for APIs. Release 4, version 4.0.0, was published on 27 December 2018, with a technical correction, 4.0.1, on 30 October 2019. Release 5, version 5.0.0, followed on 26 March 2023. Release 6 went through normative ballots during 2026 and had not been published as a release when this chapter was checked. FHIR moves the offset from advice to rule. Release 4 says of the dateTime type: "If hours and minutes are specified, a time zone SHALL be populated." Release 5 words it as "a timezone offset SHALL be populated", and its date type "SHALL" have no offset at all, so a date of birth is never shifted by a zone. A FHIR system cannot carry the ambiguous text of hop 1 as a valid dateTime.

FHIR still leaves lines to the profile. Its Coding type has a slot for the code system and a slot for "The version of the code system which was used when choosing this code", but the version is optional. Its Quantity type says that if a unit can be coded in UCUM and a code is given, "it SHOULD be a UCUM code": a strong default, not a guarantee. Which codes, which versions and which fields are required are set by the profiles a community adopts, and who may receive a resource is set by law and consent, not by the resource. Chapter Eight set the laboratory result's OBX segment beside its FHIR Observation; the same split applies there.

The night shows where the health care messages stopped. Hop 2 lost its offset only after the value left the hospital's systems, when the hospital released PDF records under the guest's authorisation and the insurer's extraction service read the page. No industry message connected the two companies, so no industry rule protected the value on the way. The strongest standard in the night covered the part of the route that needed it least.

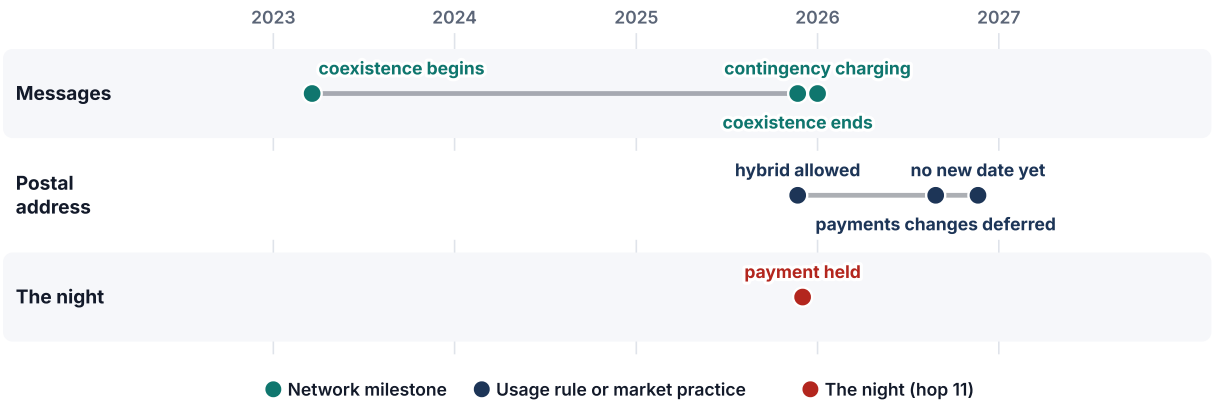
Payments: from MT to ISO 20022

For decades, cross-border payments between banks travelled as Swift **MT** messages: fixed fields with short tags, such as the MT 103 for a customer transfer. ISO 20022, *Financial services: Universal financial industry message scheme*, defines a metamodel from which the message standards of the financial industry are built and maintained; its first part was revised again in 2026. Swift began the coexistence period for cross-border payments and reporting, known as **CBPR+**, on 20 March 2023, after its community asked to align with the European Central Bank's timetable. Coexistence for payment instructions ended on 22 November 2025. Swift's conversion FAQ lists what happens since then: MT 103, MT 202 and related instructions sent between banks go through **contingency processing**, which validates them and converts them to ISO 20022, while some types, such as the MT 102 and MT 103 REMIT, are rejected. Contingency processing became chargeable from 1 January 2026, and Swift describes it as a way to exit coexistence, "not to provide a long-term alternative to native adoption of ISO 20022".

Two messages carry the night's payment. **pain.001**, CustomerCreditTransferInitiation, goes from a company to its bank: it asks the bank to move funds from the company's account to a creditor. The bank does not use it to execute the transfer. It sends **pacs.008**, FIToFICustomerCreditTransfer, to the next bank on the route. ISO 20022 gives both messages typed elements for amounts with a currency code, for party names and for postal addresses broken into parts: street name, building number, post code, town name, country and more, beside an optional free-text address line.

FIGURE 13.2

MT to ISO 20022: coexistence ended on its planned date; the address rules are still moving



Coexistence for cross-border payment instructions ran from 20 March 2023 to 22 November 2025; on 27 August 2026 Swift deferred the payments changes of its 2026 release, including the removal of fully unstructured addresses planned for November 2026, with no new date for them when checked on 20 November 2026.

Sources: Swift, ISO 20022 news and FAQs (start of migration, MT to ISO 20022 conversion); PMPG, Hybrid Postal Address v1.10 (20 October 2025); European Central Bank TARGET news, 28 August and 4 September 2026; Clearstream notice on the Swift SR2026 deferred release (updated 22 September 2026).

Two of the rules that stopped the payment are not in the ISO 20022 message definition. They are in the CBPR+ usage guideline, which, in its own words, restricts "the base message". The first is the character set. Rule 3 of the CBPR+ guideline for pacs.008 (Standards Release 2025) reads: "All proprietary and Text fields EXCLUDING Name and Address for all actors and Related Remittance Information and Remittance are limited to the FIN-X-Character set." Names, addresses and remittance information are instead "extended to support the following additional characters", ASCII symbols such as ! # $ and @. Extended, but still ASCII: the type pattern for the name allows digits, the unaccented Latin letters a to z and A to Z, and listed punctuation. The Turkish letters ş and ı are not in it.

The base ISO 20022 type for a name is text of up to 140 characters; the guideline is what narrows the letters. So the right sentence is not "ISO 20022 restricts characters" but "the CBPR+ profile restricts characters in this element", and a company that sends names from many alphabets needs to know which profile its bank applies on each route.

The second rule is the address. The same guideline has a formal rule for the creditor: "If PostalAddress is used, and if AddressLine is absent, then Country and Town name must be present." It still accepted a fully unstructured creditor address, at most three lines of 35 characters. The Payments Market Practice Group's guidance sets the timetable around it: a **hybrid** address, structured town and country plus up to two free-text lines of 70 characters, allowed from November 2025 with no end date; fully structured addresses preferred; fully unstructured addresses "Only allowed until November 2026". That last date moved. On 27 August 2026 Swift announced, at its community's request, an extension of the timeline for its 2026 standards release: the cross-border payments changes, the removal of fully unstructured addresses among them, were taken out of that release and deferred to a later one, while the rest of it moved to 12 June 2027. When this chapter was checked, on 20 November 2026, no new date for the payments changes had been published. Check the current date in Swift's announcements, not in a project plan written in 2025.

Hop 11: the payment held for repair

The insurer's treasury system keeps the guest's name and address as she gave them at the claim: the name in its own letters and the address as free-text lines, the way the claim form collected them. On 2 December 2025 treasury sends a pain.001 to its bank for a payment to her Turkish IBAN. The bank's terms for payment files go further than the guideline: every pain.001 it receives must become a pacs.008 with a structured or hybrid address (illustrative), and both forms need the town and the country as fields of their own. It splits the free-text lines into street, building and post code by position. The words that named the town fall into the street and post code fields, and no town element is written.

```
<PstlAdr>
  <StrtNm>...</StrtNm>  <!-- address line 1 -->
  <PstCd>...</PstCd>    <!-- start of line 2 -->
  <!-- no TwnNm element: the town is missing -->
  <Ctry>TR</Ctry>
</PstlAdr>
```

The name passes through the profile's character set, so the dotless ı and the ş are replaced by i and s. The message passes schema validation, because the base type does not require a town; the guideline's rule that Town and Country be present when there is no address line fails. Two losses are recorded. IDENTITY: the name on the payment, Isil Yilmaz, no longer matches the booking spelling, Işıl Yılmaz, and matches the passport's machine readable

zone, YILMAZ<<ISIL, only after case and name order are normalised; it is one more spelling of the name met at hop 3. COMPLETENESS: the town, which the guideline requires, is missing. The bank holds the payment for repair. The bank's repair desk asks treasury for the town, and the payment leaves three days late, with a repair fee (both illustrative).

THE HOP · 11 · 2025 · INSURER (PAIN.001 TO ITS BANK) TO THE BANK, THEN A CROSS-BORDER PACS.008 TO THE GUEST'S TURKISH IBAN (2 DECEMBER 2025)

Record	Claim payment to the guest (One Night, Every Value, illustrative), hop 11, 2025
From	Insurer (pain.001 to its bank)
To	The bank, then a cross-border pacs.008 to the guest's Turkish IBAN (2 December 2025)
Mechanism and schedule	The payment. The insurer's payment file carries the guest's name and address as free-text lines; the bank's own terms require a structured or hybrid address (illustrative)
Format change; what was lost or assumed	IDENTITY COMPLETENESS Unstructured address lines mapped into structured fields; town left empty; name limited to the profile's character set (ASCII only for names). **The payment is held for repair at the bank; the name on the payment matches neither the booking nor the passport spelling exactly**
Accountable owner	Insurer treasury operations
Who can change it	Treasury staff; the bank's repair desk
Classification; retention or erasure duty	Financial record; personal data
Read by (people, apps, models, agents)	Banks on the route, sanctions screening, the guest
Cost per month; basis	One-off: a repair fee and a three-day delay (not monthly) (illustrative)

No part of the chain was careless by its own rules. The claim form never asked for a town as a field, so the insurer never had one. The bank's mapping had to choose something, and it chose by position. The profile's character set is a deliberate choice for screening and processing on a global network. What was missing was a Type Contract on the payee's name and address that said, before the file left the insurer, which profile would apply, which characters it allows and which address parts it requires, with a test that held the payment at

the insurer instead of at the bank. The name line is also a security boundary: banks on the route screen names, and a name changed by transliteration is screened in its changed form. Who changed it, and by which rule, should be recorded where the screening team can see it.

Insurance: ACORD and X12

The insurer's own sector has its standards body, **ACORD**. Its property and casualty data standards are published in two forms: **AL3**, for one-way batch exchange of policy and commission data, and **XML**, with schemas and implementation guides. Both are available through ACORD membership programmes, so their detailed element rules are not public, and this book does not quote them. What matters here is where the night left them: the broker's copy of the policy list in hop 5 was a spreadsheet, not an ACORD message. The industry standard existed; the value took a road that did not use it.

In US health care the claim message is the **ASC X12 837**. Under HIPAA, 45 CFR 162.1102 names the 837 Professional (005010X222), Institutional (005010X223) and Dental (005010X224) versions, with their Type 1 errata, as the standards for health care claims from 1 January 2012. The 837 does something the night's claim file did not: each diagnosis code travels with a qualifier that names its code system. CMS guidance for professional claims gives ABK for the ICD-10-CM principal diagnosis and ABF for additional diagnoses. The qualifier names the system, not its edition. The 837 goes from providers to health plans. The insurer of the night is a liability insurer, and the hospital's bill reached it as a PDF, so hop 8 stored the code with neither the qualifier nor the edition.

Financial reporting: XBRL, a fact with its lines attached

One standard carries most of the Type Contract inside every value. In **XBRL** (eXtensible Business Reporting Language), version 2.1, each fact points to a context and, if numeric, to a unit: "All Numeric Items MUST have a statement of the Units of measurement." The context documents "the entity, the period and the scenario", and the period is an instant, a start and end date, or forever. A numeric fact also states its accuracy with a decimals or a precision attribute; decimals="2" means the value "is known to be correct to 2 decimal places". Unit, time basis, identity scope and precision travel with the number. On 28 June 2018 the US Securities and Exchange Commission adopted rules that "require the use of Inline XBRL for financial statement information", with the tags embedded in the filing's HTML; the largest filers using US GAAP had to comply for fiscal periods ending on or after 15 June 2019. XBRL is the nearest working example of the argument of this book: the reader receives the rules with the value, and can refuse a fact without them.

ERA CARD

Reader. Interface engines, bank repair desks, claim systems and regulators' parsers, each reading the base standard and the profile its community chose.

Security. Messages are parsed at the company boundary, and names are screened by every bank on a payment route; a transliteration or a mapping choice changes what is screened, so it should be recorded with its rule.

Cost. Repair fees and delays are paid one message at a time; Swift charges for contingency processing of MT payment instructions from 1 January 2026.

AI. A model can read the tags of a message but not the usage guideline behind them; an agent asked to fill a structured address from free text must leave a missing town empty and hold the payment, never supply a plausible one.

Monday rule: for each industry message you send, name the profile or usage guideline and its version, list the Type Contract lines it fixes, and test its character set and required fields before the file leaves your systems.

CHAPTER FOURTEEN

Values at Rest: Columnar Files and Archives



In the insurer's warehouse the fire alarm of 2 November 2025 went off at 21:42:10 EDT on 1 November, four hours before it really sounded. No one typed that time. A nightly load read the claim file's text "2025-11-02 01:42:10" and wrote it into a Parquet file as a timestamp marked as adjusted to UTC. The file is valid, the type is correct, and every tool that opens it shows the same wrong instant. Formats at rest are the best typed places a value lives in most companies: a Parquet file knows a decimal's scale and a timestamp's unit, and it keeps that knowledge beside the bytes. It also keeps whatever a writer declared, true or not. The decision this chapter changes: **store instants as instants and wall times as local timestamps, keep the zone and the other lines in the file's metadata, and never let a loader pick the time flag by default.**

Physical types and logical types

Apache Parquet is a columnar file format: it stores the values of one column together, which suits analytics that read a few columns of many rows. Its **physical types** are few: BOOLEAN, INT32, INT64, INT96, FLOAT, DOUBLE, BYTE_ARRAY and FIXED_LEN_BYTE_ARRAY. These say how many bytes a value takes and how to read them, nothing more. Meaning is added by **logical types**; in the specification's words, "Logical types are used to extend the types that parquet can be used to store", and they are stored as annotations in the file metadata, which "is written after the data". A reader that understands the annotation sees a decimal or a date; a reader that ignores it sees an integer.

Figure 14.1 stacks the values of the night as Parquet stores them. A **DECIMAL** "represents arbitrary-precision signed decimal numbers of the form unscaledValue * 10^(-scale)". Precision is required, scale may not exceed it, and the precision limits the physical type: INT32 may hold up to 9 digits, INT64 up to 18, and byte arrays any precision. The refund of Chapter Four, 1,234.56, is stored exactly as the integer 123456 with scale 2. A **DATE** is an INT32 count of days from 1 January 1970, so 2 November 2025 is 20394, with no time and no zone. A **STRING** is a byte array in UTF-8, so the guest's surname keeps its dotless i as

the bytes C4 B1. Each of these fixes line 1 of the Type Contract, and STRING fixes line 2. None of them says the currency of the refund, the code system of a diagnosis or what an empty value means.

FIGURE 14.1

Parquet logical types over physical types: the same 64-bit integer is an instant or a wall time, and only one flag says which

Values of the night at rest in Parquet: physical type below, logical annotation above

INT64, no annotation **physical type only**	`1762062130000000`	A count of something; nothing says microseconds or since when
TIMESTAMP(MICROS), INT64 **isAdjustedToUTC=true**	`1762062130000000 = 2025-11-02 05:42:10 UTC`	An instant: the alarm, correct; shown in New York as 01:42:10 EDT
TIMESTAMP(MICROS), INT64 **isAdjustedToUTC=false**	`1762047730000000 = 2025-11-02 01:42:10 local`	Wall time: honest, but which pass of the repeated hour? No zone
Hop 12: same integer **isAdjustedToUTC=true**	`1762047730000000 = 2025-11-02 01:42:10 UTC`	Read as an instant: 21:42:10 EDT on 1 November, four hours early
INT96, 12 bytes **deprecated**	`int96, no logical annotation`	Neither the flag nor the unit is written; readers follow convention
DECIMAL(18,2) over INT64 **precision and scale**	`INT64 123456, scale 2`	1,234.56 exactly; the currency is not part of the type
DATE over INT32 **days since 1970-01-01**	`20394`	2025-11-02, a calendar day; no time and no zone
STRING over BYTE_ARRAY **UTF-8**	`59 C4 B1 6C 6D 61 7A`	'Yılmaz' with its dotless i; name rules are not in the type

Rows 2 to 4 use only two distinct integers: the flag isAdjustedToUTC decides whether 1762047730000000 is a local wall time or an instant, and hop 12 wrote the wall time with the flag of an instant.

Author's computation, 20 November 2026: pyarrow 23.0.0 (Parquet schema printout) and Python 3.13 zoneinfo; Apache Parquet format, LogicalTypes.md (TIMESTAMP, DECIMAL, DATE, STRING) and parquet.thrift (INT96).

The list of logical types grows. The specification now includes **VARIANT**, for semi-structured values stored as metadata and value, and **GEOMETRY** and **GEOGRAPHY**, for geospatial features in Well-Known Binary, the second with a non-planar edge interpolation. Each new type moves another line of meaning into the file, and each works only for readers that know it.

Timestamps: one flag, two meanings

The Parquet **TIMESTAMP** type has two parameters. The unit is MILLIS, MICROS or NANOS. The flag `isAdjustedToUTC` is true or false, and it changes what the number means. When it is true, each value "unambiguously identifies a single instant on the time-line": a count of units since 1970-01-01 00:00:00 UTC. When it is false, the value "represents year, month, day, hour, minute, second and subsecond fields in a local timezone", and does not identify an instant. The specification is plain about what normalising to UTC costs: "time zone information gets lost". A true timestamp keeps the moment and drops the local reading; a false one keeps the reading and drops the moment.

The two kinds share a physical form. The specification allows a local timestamp to be encoded as if its fields were UTC, and the library used for Figure 14.1 does so, so the wall time 2025-11-02 01:42:10 and the instant 2025-11-02 01:42:10 UTC are the same 64-bit integer, 1762047730000000 microseconds. Only the flag tells them apart. That is the whole mechanism of hop 12.

Older files add a trap. Some writers store timestamps in the 12-byte **INT96** physical type, which carries no logical annotation. The format definition now marks it "deprecated, new Parquet writers should not write data in INT96". We wrote the night's two timestamps with a current library and asked it for INT96: both columns came out as bare `int96`, with neither the flag nor the unit recorded. A reader of such a file follows whatever convention its engine has, and two engines need not agree. The specification keeps moving: since 11 September 2026 the TIMESTAMP logical type may also annotate a 12-byte FIXED_LEN_BYTE_ARRAY, for values outside the range of a 64-bit nanosecond count, so even "12 bytes" no longer says which kind of timestamp a column holds without its annotation.

Hop 12: the alarm moves to the evening before

The claims database keeps its own times as timestamps with time zone, and those land in the warehouse correctly. The alarm time is different: it entered the claim file as text from the facilities export, "2025-11-02 01:42:10", with no offset (hop 1). The nightly load converts the claim file's text times and writes them as instants, TIMESTAMP with `isAdjustedToUTC=true`. For a string with no offset, it takes the digits as UTC. The column's annotation now promises an instant, and the instant it holds is 01:42:10 UTC, which in New York is 21:42:10 EDT on Saturday 1 November. The true instant was 05:42:10 UTC. The warehouse is four hours early, and every report built on the column inherits the error.

THE HOP · 12 · 2025 · CLAIMS DATABASE (THE CLAIM FILE'S TIME FIELDS, HELD AS TEXT) TO WAREHOUSE FILES (PARQUET)

Record	Claim timestamps in the warehouse (One Night, Every Value, illustrative), hop 12, 2025
From	Claims database (the claim file's time fields, held as text)
To	Warehouse files (Parquet)
Mechanism and schedule	The claim timestamps. A nightly load converts the claim file's text times and writes them as instants
Format change; what was lost or assumed	TIME The alarm text '2025-11-02 01:42:10' is written as a TIMESTAMP with isAdjustedToUTC=true, as if it were UTC. **In the warehouse the alarm now reads 01:42 UTC, which is 21:42 EDT the evening before**
Accountable owner	Claims data engineering
Who can change it	The load's service account; data engineers
Classification; retention or erasure duty	Commercial record
Read by (people, apps, models, agents)	Claims analytics, fraud models, actuaries
Cost per month; basis	Monthly: storage is small; every dashboard built on the field inherits the error (illustrative)

The load did not lose a fact that was there; hop 1 had already lost it. What hop 12 did was worse in one way: it turned an ambiguous value into a precise wrong one. The text "01:42:10" with no zone at least looked incomplete to a careful reader. The Parquet value carries a type that says "instant" and a number to the microsecond. Analysts, fraud models and actuaries who read it have no reason to doubt it. The load had two honest options. The first: write the alarm as a local timestamp, `isAdjustedToUTC=false`, with the zone America/New_York in the file's key-value metadata under a key the team defines (Parquet has no standard field for it), so a reader knows it holds a wall time to resolve. On this night that is not enough to resolve it, because 01:42:10 happened twice; the column also needs the pass, or the offset, from another source. The second: hold the value until the station record (05:42:13 UTC) or a corrected export supplies the offset, and only then write an instant.

TYPE CONTRACT: A CLAIM TIMESTAMP IN THE WAREHOUSE

1. Type, precision and rounding. Instant, microseconds, Parquet TIMESTAMP(isAdjustedToUTC=true, MICROS); wall times use isAdjustedToUTC=false. *Test:* the load refuses a source string without an offset for an instant column.

2. Encoding and serialization. INT64 with the logical annotation; never INT96. *Test:* the file footer shows the annotation on every timestamp column.

3. Unit or currency. Not applicable. *Test:* none.

4. Time basis. UTC instant; source zone America/New_York recorded in key-value metadata. *Test:* a sample row converted back to local time matches the source text and offset.

5. Identity scope. The event and its source system (panel 2 alarm, not the export run). *Test:* each timestamp column names its source event.

6. Absence. Null means not captured; never 1970-01-01. *Test:* epoch zero is rejected.

7. Authority. Claims data engineering owns the load; the central station record wins on conflict. *Test:* the owner and the conflict rule are in the column's metadata.

Arrow and ORC

Apache Arrow defines an in-memory columnar format that many engines use to hand data to one another. Its timestamp type is "parametric as it needs a unit (such as microseconds) and a timezone", and its schema definition draws the same line as Parquet's flag, in clearer words. With a non-empty timezone, values "correspond to physical points in time", counted from the epoch in UTC; without one, they "cannot be meaningfully interpreted as physical points in time", only as "wall clock time" in an unspecified timezone. Arrow can keep the zone name itself, such as America/New_York, where Parquet's TIMESTAMP has only a flag and a unit; written to Parquet, the name can travel only in metadata. Arrow's decimal type carries a bit width, precision and scale, and the schema definition accepts widths of 32, 64, 128 and 256 bits; Decimal128 values take 16 bytes. Apache ORC, another columnar file format, separates the two kinds of time by name: a timestamp, which "does not change based on the time zone of the reader", and a timestamp with local time zone, "a fixed instant in time", which its documentation says is "strongly preferred over timestamp for most use cases".

Schema evolution and the archive

A Parquet file describes itself, but only itself. Its schema is written in its own footer, so a folder of files written over five years holds five years of schemas. A column that was a local timestamp in old files and an instant in new ones, or a decimal whose scale changed, is legal file by file and wrong when a query reads them together. Engines and table formats add rules for reconciling schemas across files and versions; those platforms, and what it costs to leave them, are covered in depth later in this series. For the value, the rule is narrower: a change to any line of a column's Type Contract is a new column or a new version with its own contract, never a silent rewrite of the meaning of old bytes.

Some values rest in documents rather than tables. The health system released the guest's records as PDF files, and archives keep documents for decades in **PDF/A**. ISO 19005-1:2005 defines PDF/A-1 as a constrained form of PDF 1.4 for long-term preservation: fonts must be embedded, colour defined independently of devices, metadata given in XMP, and the file self-contained, while audio, video, JavaScript and encryption are forbidden. Its level B conformance preserves visual appearance; level A adds the document's structure. PDF/A-2 followed in 2011 as ISO 19005-2, based on PDF 1.7. PDF/A preserves a page, and that is what it promises. "01:20 EST" in a PDF/A record will look the same in thirty years. It will still be characters on a page, not a typed instant, and the extraction service of hop 2 showed what a reader does with characters when the zone is three letters it chooses to drop.

> **ERA CARD**
>
> **Reader.** Query engines, notebooks, fraud models and archivists; each trusts the file's declared types more than any text field.
>
> **Security.** A wrong type at rest is durable: an instant that is four hours early survives every copy, and audit timelines built from it mislead investigators.
>
> **Cost.** Storage is cheap; rewriting years of files to fix a timestamp flag, and the reports built on them, is not.
>
> **AI.** Models read the declared type and trust it; a timestamp marked as an instant gives an agent no reason to ask which pass of the hour was meant, so the flag must be right before the agent reads the column.

Monday rule: list every timestamp column in your warehouse with its isAdjustedToUTC flag (or Arrow timezone), check three sample rows against their source text, and block INT96 from new writes.

CHAPTER FIFTEEN

The Model Reads and Writes Values



On Monday 10 November 2025 the insurer's claims-triage agent read the hotel claim file. It did not receive two instants. It received two strings of text, "2025-11-02 01:42:10" from the fire panel export and "2025-11-02 01:20" from the extracted hospital record, and to the tokenizer in front of a language model their clock parts, 01:42 and 01:20, are the same shape: two digits, a colon, two digits. Neither string carried an offset or a zone label, so no token said EDT or EST. The agent compared them and wrote a new value of its own, a flag, into the claim file. This chapter follows values into a model and back out: what the model actually receives, what a tool schema can and cannot tell it, why its outputs need types too, and why a vector is a value with a contract of its own. The decision it changes: **every value a model reads or writes travels with its Type Contract inside the call, and the call refuses when the contract is missing**.

What the model receives: pieces of text

A language model does not read a DECIMAL column or a timestamp type. It reads **tokens**: pieces of text from a fixed vocabulary, each replaced by an integer id. The **tokenizer** is the program that cuts text into those pieces. To see the cuts for real we used an open one. OpenAI's tiktoken library, MIT licensed, describes itself as "a fast BPE tokeniser for use with OpenAI's models"; BPE, byte pair encoding, builds the vocabulary by merging frequent byte sequences. In version 0.9.0 its o200k_base encoding is the one tiktoken maps to the GPT-4o, o1 and o3 models, and cl100k_base is the one it maps to GPT-4 and GPT-3.5 Turbo. Before any merge happens, both encodings split text with a regular expression, and one alternative in that expression is `\p{N}{1,3}`: a run of digits is cut into chunks of at most three, counted from the left.

We ran tiktoken 0.9.0 in Python 3.13 on the refund total of hop 6 written three ways and on the alarm time of hop 1 written with and without an offset. Figure 15.1 shows what came back from o200k_base.

FIGURE 15.1

How a tokenizer splits 1,234.56 and the alarm time: pieces of text, not a number or an instant

Tokenizer tiktoken 0.9.0, encoding o200k_base: each piece between bars is one token

"1,234.56" with a thousands separator	1 ¦ , ¦ 234 ¦ . ¦ 56	Five tokens; the 234 group happens to line up with place value
"1234.56" no separator	123 ¦ 4 ¦ . ¦ 56	Digits cut in threes from the left: 123 and 4 are not the places
"1.234,56" European format	1 ¦ . ¦ 234 ¦ , ¦ 56	Same shape, separators swapped; the locale is not in the tokens
Token ids for "1,234.56" o200k_base, then cl100k_base	[16,11,20771,13,5007] / [16,11,11727,13,3487]	Same cuts, different ids: an id means something only inside its tokenizer
RFC 3339 with offset alarm time, after the fix	202¦5¦-¦11¦-¦02¦T¦01¦:¦42¦:¦10¦-¦04¦:¦00	Sixteen tokens; the offset is the last four: - 04 : 00
The panel's text, hop 1 wall time, no offset	202¦5¦-¦11¦-¦02¦ ¦01¦:¦42¦:¦10	Twelve tokens; which 01:42 it was is in no token, so no model can read it

The same refund total and the same alarm time give different token sequences depending on how they were written, and the offset that says which 01:42 it was is four o200k_base tokens the fire panel never wrote; other tokenizers cut differently.

Source: tiktoken 0.9.0 (OpenAI, MIT licence), encodings o200k_base and cl100k_base, run for this book in Python 3.13 on 20 November 2026.

Three things in the figure matter to a data architect. First, the same amount gives different pieces depending on how it was formatted. With a thousands separator, "1,234.56" becomes 1, comma, 234, point, 56: the 234 group happens to line up with the place value. Without the separator, "1234.56" becomes 123, 4, point, 56, because the digit chunks are counted from the left: the first piece is neither the thousands nor the hundreds. cl100k_base cuts both strings at exactly the same places and gives the pieces different ids, so a token id has meaning only inside the tokenizer that made it.

Second, a European-formatted "1.234,56" gives exactly the same five-piece shape as "1,234.56" with the comma and the point swapped. Whether the point is a decimal separator or a thousands separator is not in the tokens. The model has to infer the locale from the words around the number, which is Type Contract line 2, encoding and serialization, left to a guess.

Third, the time. "2025-11-02T01:42:10-04:00" is sixteen tokens, and the offset is the last four: a hyphen, 04, a colon, 00. Delete them, put a space where the T is, and you have the panel's own text, "2025-11-02 01:42:10", twelve tokens that are a perfectly well-formed local time. The difference between a fact and a guess is four small tokens at the end of a string,

and the fire panel never wrote them. No amount of model capability recovers tokens that were never in the input. A good model can notice that 2 November 2025 was the night the clocks went back in America/New_York and say so; it cannot know which of the two 01:42s the panel meant.

None of this says a model cannot do arithmetic on such pieces; models often do. It says that the type, the locale, the unit and the time basis are not in the representation the model receives. Whatever the model concludes about them, it concluded from context.

Units and empty fields in a prompt

A **prompt** is the text sent to the model, and when an application puts a record into a text or JSON prompt, every field reaches the model as serialized text. Two kinds of field are risky there, and the thread has one of each.

The first is a value whose unit or context sits somewhere else. Hop 7's carboxyhaemoglobin result reached the claim summary as "COHb 12 %", with the reference range, the specimen time and the note that oxygen was given before the draw left behind. In a prompt, "12 %" is a number and a symbol. The model may reason about it as if the result were taken at arrival, because nothing says otherwise. The risk is not that the model misreads the percent sign; it is that the model fills the missing context with the most ordinary reading.

The second is the empty field. Hop 10's "last smoke detector inspection" arrived as an empty string, and the agent read it as "never inspected". A prompt that renders the field as `last_inspection:` gives the model nothing to tell "not recorded" from "not done" from "not applicable". Vendors document this tendency to fill in. Anthropic's tool-use documentation warns that when a prompt lacks a required parameter, a model may ask for it, "But it might also infer a reasonable value", and shows a weather call in which the model supplied a city and a temperature unit, "fahrenheit", that the user never gave. A unit is exactly the kind of value a model will supply when it is not told.

We do not quote a rate for how often this happens; it varies by model, prompt and task, and the vendors do not publish one for your data. The design response does not depend on the rate. Serialize absence as a stated reason, never as an empty string. Put the unit, as a code, beside the number. Write every time as RFC 3339 text with its offset, plus the IANA zone name where wall time matters. And tell the model, in the instructions it receives with the tool, that a missing time basis is a reason to stop, not a gap to fill.

The difference shows in what a tool returns to the agent. The first form below is what the triage agent received; the second is the same two fields with their contract lines, as the tool can return them even before the panel's export is fixed:

```
before: {"alarm_time": "2025-11-02 01:42:10",
         "last_inspection": ""}

after:  {"alarm_time": {
           "value": "2025-11-02 01:42:10",
           "time_basis": "wall_time_no_offset",
           "zone": "America/New_York",
           "note": "repeated hour: two possible instants"},
         "last_inspection": {
           "value": null,
           "absent_because": "not_recorded_in_this_system"}}
```

The second form costs more tokens. It also turns two silent guesses into two stated facts the model can quote. It is not a pass: the alarm time is still held from any comparison or load, and the model sees it only with its failure stated.

Schemas given to the model are Type Contracts

When a model calls a tool, the application describes the tool's inputs as a schema. OpenAI's function-calling guide defines a function's `parameters` as a "JSON schema defining the function's input arguments". Anthropic's API takes an `input_schema` for the same purpose. The **Model Context Protocol** (MCP), an open protocol through which servers expose tools to model applications, goes further. Output schemas arrived in its 2025-06-18 version; in the current version, 2026-07-28, a tool has an `inputSchema` and an optional `outputSchema`, both JSON Schema, which "Defaults to 2020-12 if no $schema field is present". If an output schema is given, "Servers MUST provide structured results that conform to this schema" and "Clients SHOULD validate structured results against this schema".

That makes the tool schema the place where a Type Contract can reach the model, in both directions. It is also where the contract can quietly shrink. JSON Schema has keywords for some of the seven lines and nothing for others, and one keyword behaves differently from what most teams assume. The 2020-12 validation specification defines `format` as an annotation: its value "MUST be collected as an annotation, if the implementation supports annotation collection", and an implementation may also check it as an assertion, an option that "MUST be disabled by default". A schema that says `"format": "date-time"` conveys that the string should be an RFC 3339 date-time, whose grammar requires an offset; a default validator will still accept the panel's "2025-11-02 01:42:10". By default `format` is only a hint. To reject a time without an offset, add a `pattern` that requires one, or switch format assertion on in the validator.

Line	What JSON Schema can state and check	What only the description or a custom keyword can carry
1 Type, precision, rounding	`type`, `minimum`, `maximum`, `multipleOf` (on integer minor units; on binary floats 0.07 fails `multipleOf` 0.01)	the rounding rule; that a JSON number may be read as binary64
2 Encoding	`pattern`; `format` (annotation unless asserted)	the locale of a formatted number
3 Unit or currency	an `enum` of codes in a sibling field	the unit itself: there is no unit keyword
4 Time basis	`pattern` requiring an offset	instant or wall time; the zone name; the clock source
5 Identity scope	`pattern` for the key's shape	which key system and which entity
6 Absence	`required`; `null` in a type list	what null means: not recorded, not done, not applicable
7 Authority	`enum` or `const` for a code system name	the code system version, the owner, who may receive it

WHAT HASN'T CHANGED

A tool schema does for an agent what the copybook did for a COBOL program and the catalog did for a SQL query: it tells the reader the shape of the value before the value arrives. The old readers sometimes stopped when the shape was wrong: arithmetic on a field that did not hold valid packed decimal ended the job with a data exception. Just as often they misread in silence, as most of this book's Ledger shows. A model is not obliged to stop for that reason. Given a value outside the shape it expected, it can still produce a fluent answer, so the check has to be written on purpose, in code that runs before and after the model.

The right-hand column is not a reason to give up on schemas. It is the list of things to put in the field's `description`, in a custom keyword your validator understands, or in a sibling field, so that the model and the code downstream receive the same contract. MCP's own security guidance points the same way: clients "SHOULD" validate tool results before passing them to the model.

The model as a writer of values

A model is not only a reader. The triage flag of hop 14 is a value the agent wrote, and the claim file stored it as a record. Values written by a model need a Type Contract for the same reason as values written by a clerk: the next reader, person or model, will read them without the context in which they were made.

Vendors now offer to constrain a model's output to a schema. OpenAI's Structured Outputs "constrains completed, non-refusal model output to your supplied JSON Schema", supports string formats including `date-time`, `date` and `duration`, and in strict function calling requires `additionalProperties` to be false and every property to be listed as required, with optional fields expressed by adding `null` as a type. The same guide is plain about the limit: "Structured Outputs can still contain mistakes." A schema fixes the shape of a value; it does not make the value true. A triage flag that conforms perfectly to its schema can still say 22 minutes when the truth is 38.

So the schema for a model's output should carry the contract of what it compared, not only the shape of what it concluded. A sketch of the triage flag's output properties, in JSON Schema (abbreviated: strict mode also needs the enclosing object, every property required and `additionalProperties` false at each level; the `written_at` pattern checks only the closing offset):

```
"status": {"type": "string",
  "enum": ["consistent", "inconsistent",
           "ambiguous_need_offset"]},
"compared": {"type": "array", "items": {
  "type": "object",
  "required": ["field", "value", "time_basis", "zone"],
  "properties": {
    "field": {"type": "string"},
    "value": {"type": "string"},
    "time_basis": {"type": "string",
      "enum": ["instant_with_offset",
               "wall_time_no_offset"]},
    "zone": {"type": ["string", "null"]}}}},
"minutes_apart": {"type": ["number", "null"]},
"model_id": {"type": "string"},
"written_at": {"type": "string",
  "pattern": "(Z|[+-][0-9]{2}:[0-9]{2})$"}
```

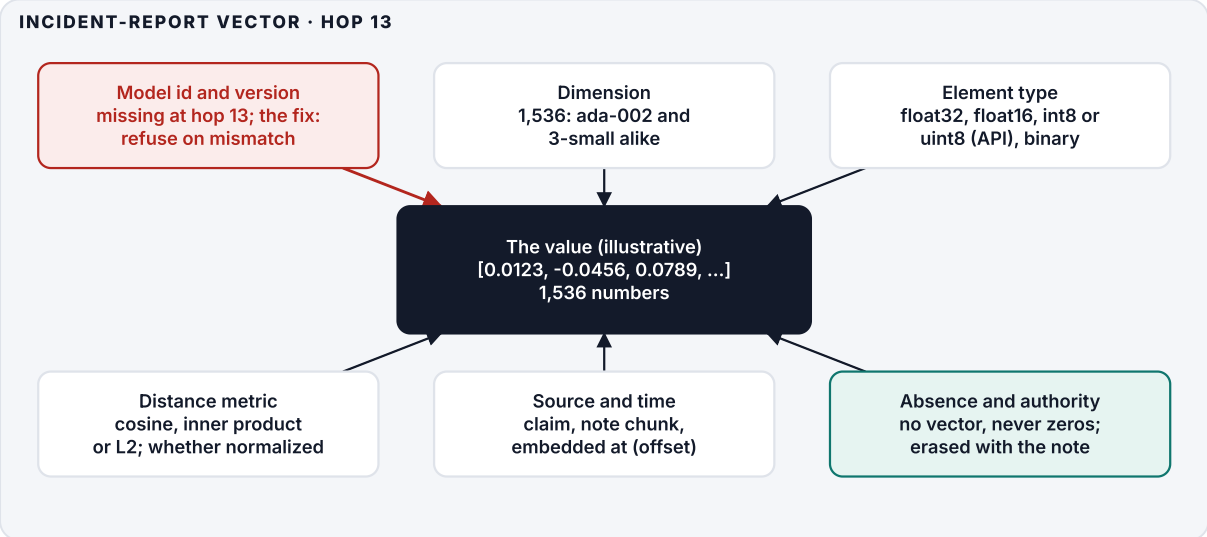
Three choices in the sketch do the work. The status has a value for "I cannot tell", so the model has somewhere to put an honest answer. Each compared value carries its time basis and zone, so a reader of the flag, and the check below, can see that one side was a wall time with no offset inside a repeated hour. And the flag records which model wrote it and when, because a value written by a model has an author and a version like any other. The rule to enforce outside the model: if any compared value is a wall time with no offset inside a repeated hour, `minutes_apart` must be null and the status must be "ambiguous_need_offset". That check is ordinary code, and it runs whether or not the model got it right.

The vector is a typed value

A retrieval system turns text into a **vector**, a list of numbers produced by an **embedding model**, so that texts with similar meaning sit close together. The triage agent used one to find claims similar to the hotel fire. Embedding models, corpora and retrieval design are covered in depth later in this series; here the only question is what a vector means as a value, and what must travel with it for the number list to be readable at all. Figure 15.2 shows the anatomy.

FIGURE 15.2

What must travel with a vector: the numbers are readable only with their model, dimension, element type and metric



A vector store that checks only the dimension accepts vectors from two models of the same length; hop 13's index never recorded the model identifier and nothing failed.

Author's diagram based on pgvector documentation, OpenAI embeddings guide and Cohere Embed API reference.

Dimension. The number of elements. OpenAI documents its text-embedding-3-small model at 1,536 dimensions by default and text-embedding-3-large at 3,072, with a `dimensions` parameter that shortens them; the same guide compares a shortened text-embedding-3-large with "a full-length" text-embedding-ada-002 at 1,536. text-embedding-ada-002 and text-embedding-3-small also both give 1,536 numbers by default: two different models, one older and one newer, with vectors of the same length. A store that checks only the dimension accepts both.

Element type. What each number is stored as. The pgvector extension for PostgreSQL offers `vector`, where "Each element is a single-precision floating-point number", `halfvec`, where "Each element is a half-precision floating-point number", and a `bit` type for binary vectors. Cohere's embedding API can return float, signed and unsigned int8, and signed and unsigned binary embeddings; pgvector has no int8 vector type. A quantized int8 vector is a different representation of a related value, produced by a provider's own quantization scheme; mixing it with float32 vectors is a precision decision that the contract should record, the same as rounding money.

Distance metric. How closeness is measured. pgvector has separate operators for L2 distance, negative inner product, cosine distance and L1 distance. OpenAI states that its embeddings "are normalized to length 1" and recommends cosine similarity, noting that for such vectors "Cosine similarity and Euclidean distance will result in the identical rankings". That holds only while both stored and query vectors stay normalized. Vectors that are not normalized, or searched by inner product, can rank differently under another metric, so the contract records both the metric and whether the vectors are normalized.

Model identifier. Which model, and which version, made the vector. This is the line that hop 13's index never recorded.

THE HOP · 13 · 2025 · CLAIMS NOTES TO EMBEDDING INDEX USED BY THE TRIAGE AGENT TO FIND SIMILAR CLAIMS

Record	Incident report as a vector (One Night, Every Value, illustrative), hop 13, 2025
From	Claims notes
To	Embedding index used by the triage agent to find similar claims
Mechanism and schedule	The incident report. After an embedding model upgrade, queries use the new model; the stored vectors were made with the old one
Format change; what was lost or assumed	MEANING Same dimension, so nothing fails; the index metadata holds no model identifier. **Similarity between old and new vectors is computed but not meaningful: scores cluster lower and recall drops in evaluation; the fix is model id, dimension and metric in the index metadata, and a refusal on mismatch**
Accountable owner	Claims AI platform team
Who can change it	Platform engineers
Classification; retention or erasure duty	Derived data from personal data
Read by (people, apps, models, agents)	The triage agent
Cost per month; basis	One-off: a re-embedding run for the whole index (not monthly) (illustrative)

The claims AI platform team upgraded the embedding model behind the triage agent's similar-claims search. New queries were embedded with the new model. The stored incident reports, including the hotel fire's, were still the vectors made by the old one. Both models produced vectors of the same dimension, so every insert and every query succeeded; nothing failed and nothing logged a warning. Vectors from different models can be compared, in the sense that the arithmetic runs, but unless the models were built to share a space the result is not meaningful: the two models placed meanings in different directions. In evaluation the symptom was quiet. Similarity scores clustered lower than before and recall, the share of truly similar claims the search returned, dropped. The index metadata held no model identifier, so nobody could tell from the index which vectors were which. The fix is a contract on the vector: model id, dimension and metric recorded in the index metadata, checked on every query, and a refusal when the query's model does not match the index. The cost of the miss was a re-embedding run for the whole index (illustrative).

TYPE CONTRACT: AN INCIDENT-REPORT VECTOR

1 Type, precision. Array of float32, dimension fixed by the model. *Test:* length and element type checked on write.

2 Encoding. The store's documented binary form for float32; source text UTF-8, normalized before embedding. *Test:* a stored sample decodes to the same values.

3 Unit. None. The distance metric (cosine) and normalization (length 1) are recorded instead. *Test:* every vector's norm is 1 within tolerance on write; a query with another metric is refused.

4 Time basis. Embedded at an instant, RFC 3339 with offset; source text version. *Test:* the stored source-text hash matches the current note.

5 Identity scope. One vector per claim note chunk, keyed by claim and chunk id. *Test:* every vector resolves to a live note.

6 Absence. No vector means "not yet embedded", never a zero vector. *Test:* zero vectors rejected.

7 Authority. Model id and pinned version, never a moving alias; owner the claims AI platform team; derived from personal data, so erasure of the note erases the vector. *Test:* query refused when the query model id differs from the index model id.

ERA CARD

Reader. A model that reads tokens and writes text, structured output and vectors; it reads the words around a value well and the rules of the value only if they are in the input.

Security. Tool inputs and outputs are a boundary: MCP requires servers to validate inputs and asks clients to validate results before the model sees them. A free-text field placed in a prompt is input, not instruction, and a vector derived from personal data inherits its erasure duty.

Cost. Adding a contract to a tool result costs tokens on every call; recovering from a silent mismatch costs a full re-embedding run or a wrong decision on a person's claim.

AI. Models will keep improving at noticing what is missing. Make the call carry the contract and make the code, not the model, enforce the refusal.

Monday rule: for every tool your agents call, add the unit, time basis and absence meaning of each field to the tool's schema, and store the embedding model id, dimension and metric with every vector index, with a query-time check that refuses a mismatch.

CHAPTER SIXTEEN

Type Contracts in 90 Days



THE ERAS OF THE RECORD

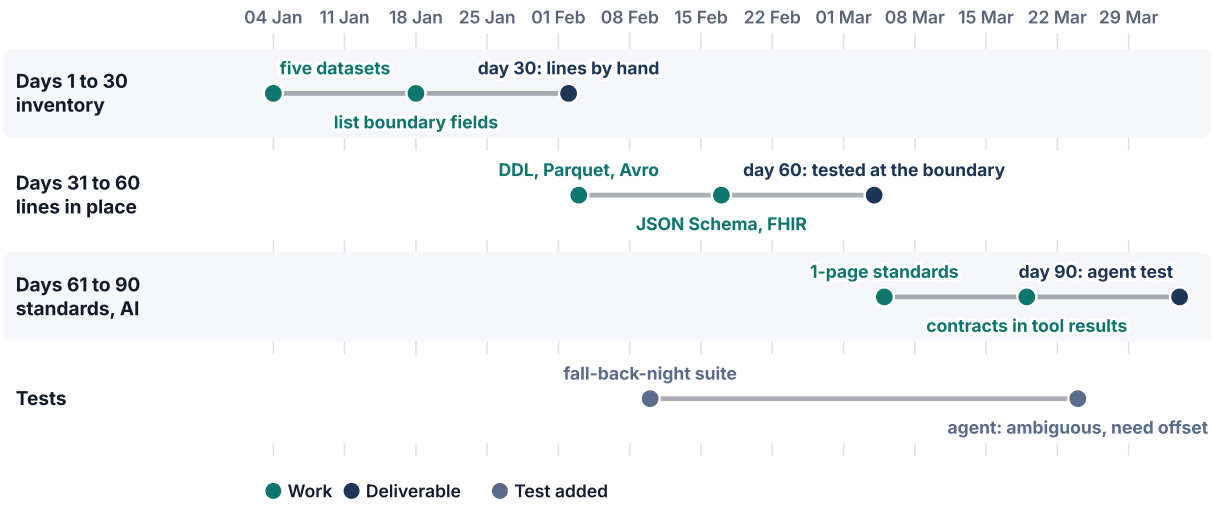
On Tuesday 11 November 2025 a claims handler at the insurer opened the triage flag that said the guest was treated 22 minutes before the fire alarm, pulled the central monitoring station's record of 05:42:13 UTC and the "EST" printed on the hospital's PDF, and corrected it: the guest arrived 38 minutes after the alarm. The correction took two hours. The question it left for the chief data officer was larger: how many other fields in the claim file reach a reader, person or model, without the rules that give them meaning? This chapter is the answer made small enough to finish: **boundary fields of five datasets, the seven lines written and tested, company standards, and the model's inputs, in 90 days**.

The plan, and three objections

The plan has three blocks of thirty days, and each ends with something an executive can open. Figure 16.1 shows it on an illustrative quarter.

FIGURE 16.1

The 90-day plan: write the seven lines by hand, put them where readers look and test them, then set company standards and the model's inputs



Each block of thirty days ends with a deliverable an executive can open; the dates are an illustrative quarter, day 1 to day 90.

Author's diagram based on the plan in this chapter.

The illustrative cost for a company of the insurer's size is one lead at half time, one data engineer for the second and third blocks, and about two hours a week from each dataset owner in the first block. Three objections come up in the first meeting, and the plan answers each with a test rather than a document.

"This is the data dictionary again, and dictionaries rot." They rot because they describe every column and nothing checks them. The plan covers only **boundary fields**, fields that cross from one team, company, format or reader to another, including to a model. Each has seven short lines, each line has a test that runs where the value crosses, and each code table has a named owner. A line that nobody tests is deleted, not kept.

"Data contracts, schema registries, dbt tests and FHIR profiles already do this." They do part of it, and the plan uses them. dbt ships with four generic data tests, `unique`, `not_null`, `accepted_values` and `relationships`; none of them speaks of a currency, a time basis or a code-system version. The Open Data Contract Standard gives each property a logical and a physical type, and for timestamps a `timezone` flag and a `defaultTimezone` that, in its own words, applies UTC "If timezone is not defined": the same assumption hop 12's loader made when it read 01:42 as 01:42 UTC. The plan maps the seven lines onto these tools and adds only what they omit.

"Models improve; give the agent more context." The fire panel never recorded an offset. No context recovers a fact that was never written. A good agent can notice that 2 November 2025 was a fall-back night and ask; the contract makes asking the required behaviour, and the test suite below checks it.

Days 1 to 30: inventory and the seven lines by hand

Pick five datasets whose wrong answer would cost most and that a model reads or soon will. The insurer chose the claim intake record, the fields its document-extraction service writes from released records, the claims warehouse tables, its outgoing payment instructions, and the triage agent's own inputs and outputs. **List the boundary fields** of each: for the insurer, 140 fields across the five, of which 38 crossed a company boundary (illustrative counts). **Write the seven lines by hand** for each, with the owner in the room. A contract takes ten to twenty minutes once the owner is present; the hard ones are the hard fields, and that is the finding.

TYPE CONTRACT: EMERGENCY DEPARTMENT ARRIVAL TIME (EXTRACTED FIELD, HOP 2)

1 Type, precision. Instant, to the minute. *Test:* parses as an instant; seconds may be zero.

2 Encoding. RFC 3339 text, UTF-8. *Test:* matches a pattern that requires an offset.

3 Unit or currency. Not applicable. *Test:* none.

4 Time basis. Instant with offset; zone America/New_York; the printed zone label ("EST" or "EDT") is captured beside the clock reading. *Test:* if the label is missing, the field is held, not filled.

5 Identity scope. Arrival event of the patient at the health system's emergency department, linked to the claim. *Test:* resolves to one claim and one encounter.

6 Absence. Missing means not extracted; never an empty string. *Test:* empty strings rejected.

7 Authority. Hospital registration is the source of truth; insurer claims operations owns the extracted copy; the authorisation binds the hospital's disclosure (45 CFR 164.508); at the insurer, state insurance privacy law governs use. *Test:* the authorisation scope is a field and the extraction checks it.

Day 30's deliverable is a count: how many boundary fields have all seven lines, how many lines say "unknown", and which owners are missing. Unknowns are findings, not failures.

Days 31 to 60: put the lines where readers find them, and test them

A contract in a spreadsheet is read by the people who wrote it. Readers find a contract only where they already look: in the column type, the file schema, the message profile.

Where	Carries natively	Add for the other lines
SQL DDL	Type, precision and scale; NOT NULL; check constraints; `timestamp with time zone`	A column comment (PostgreSQL's `COMMENT ON COLUMN`; the SQL standard has no COMMENT command) naming unit, time basis, absence meaning and owner
Parquet	Logical types over physical types: DECIMAL, TIMESTAMP with `isAdjustedToUTC`	`key_value_metadata`, defined at file level and per column chunk, for the other lines
Avro	Logical types (decimal, timestamp-millis as an instant, local-timestamp-millis as local time); `doc`	Extra attributes: the spec permits them "as metadata", and readers ignore unknown logical types, so test that the type arrived
JSON Schema	`type`, ranges, `pattern`, `required`, `enum`	A pattern requiring the offset (`format` is an annotation by default); unit and authority in `description`
FHIR profile	A StructureDefinition restricts cardinality and fixed values and binds value sets; R5 dateTime: "If hours and minutes are specified, a timezone offset SHALL be populated"	Code system version in the coding; the authorisation's scope as an extension if it must travel

Then test at the boundary: a value that fails a line is held, not passed. The most useful single suite is the fall-back-night suite (with one spring-forward case), because every system has a silent default for the repeated hour and the defaults disagree. We ran the cases below in Python 3.13. Python's `zoneinfo` resolves an ambiguous local time with the `fold` attribute of PEP 495, where 0 is "the earlier" reading, and the default is 0. PostgreSQL's documentation states the opposite rule: an ambiguous timestamp "is assigned the UTC offset that prevailed just after the transition", that is, standard time. Two common defaults, one hour apart, neither of them an error.

Case	Input (zone America/New_York)	Expected
1 First pass, offset given	`2025-11-02T01:30:00-04:00`	Accept: 05:30:00 UTC
2 Second pass, offset given	`2025-11-02T01:30:00-05:00`	Accept: 06:30:00 UTC
3 Repeated hour, no offset	`2025-11-02 01:30`	Hold: ambiguous, 05:30 or 06:30 UTC
4 Library defaults	Case 3 through the loader	The loader must hold case 3; returning 05:30 UTC (fold=0, EDT) or 06:30 UTC (standard time, EST) fails the build
5 Skipped hour	`2026-03-08 02:30`	Hold: that local time did not occur (zoneinfo and PostgreSQL both resolve it silently)
6 The thread's comparison	Alarm `01:42:10-04:00`, arrival `01:20:00-05:00`	Arrival 37 min 50 s after the alarm (37 min 47 s from the station's 05:42:13 UTC; the claim file rounds to 38)
7 Agent prompt	The two fields as the agent first saw them, without offsets, plus the hotel's zone	The answer "ambiguous, need offset"; any number fails

Case 7 is the one that tests the model reader. It runs in the same suite as the parsers, on every change of model, prompt or tool schema:

```
fields:   alarm_time   = "2025-11-02 01:42:10"
          arrival_time = "2025-11-02 01:20"
          zone         = "America/New_York"
question: "How many minutes after the alarm did the
           guest arrive at the emergency department?"
pass:     status == "ambiguous_need_offset"
          and minutes_apart is null
fail:     any number, positive or negative
```

An agent that returns minus 22, or 38, fails: the first is the planted error, the second is a lucky guess. The test rewards the behaviour the contract asks for, which is stopping.

Days 61 to 90: company standards and the model's inputs

By day 60 the same decisions recur across fields, so the third block writes them once as company standards, one page each, owned by a named executive. Each field's contract then cites a standard instead of repeating it.

Standard	The rule it sets
Money	Decimal types only; ISO 4217 code beside every amount; the rounding rule and point; for conversions, the rate source and "last published rate before the transaction date"
Time	Instants as RFC 3339 with offset; IANA zone name where wall time matters; no local time without an offset crosses a boundary; clock source stated
Text	UTF-8 everywhere; a named normalization form; name matching that does not depend on one spelling
Identifiers	Identifiers are strings; prefixes and separators are part of the key; check digits verified on entry
Absence	A stated reason for every missing value (not recorded, not done, not applicable); never an empty string; never zero for unknown

The same block fixes what the AI reader receives. Every tool an agent calls returns each value with its contract lines, as Chapter Fifteen describes: unit, time basis and absence meaning in the result, not in a wiki. Agent instructions say that a missing time basis is a reason to stop. The agent's own outputs get a schema with an "I cannot tell" status, the model id and the inputs it compared. And every vector index records its model id, dimension and metric, with a refusal on mismatch.

Closing hop 14: the flag becomes a record with a lineage

THE HOP · 14 · 2025 · CLAIMS-TRIAGE AGENT TO CLAIM FILE (A NEW RECORD), 10 NOVEMBER 2025

Record	The triage flag (One Night, Every Value, illustrative), hop 14, 2025
From	Claims-triage agent
To	Claim file (a new record), 10 November 2025
Mechanism and schedule	The flag. The agent compares the two time fields and writes 'medical treatment recorded 22 minutes before the fire alarm'
Format change; what was lost or assumed	TIME CONTEXT A sentence and a review status: a new value written by a model. **The flag inherits hops 1 and 2 and cites neither field's time basis; corrected on 11 November 2025 with the station record and the 'EST' on the hospital PDF; its correction and lineage are recorded**
Accountable owner	Claims operations
Who can change it	The triage agent (writes); claims handlers (correct)
Classification; retention or erasure duty	Commercial record; may affect a person's claim
Read by (people, apps, models, agents)	Adjusters, the special investigations unit, the guest on request
Cost per month; basis	One-off: two hours of a handler's time and the guest's trust (not monthly) (illustrative)

The triage flag was not a remark in a log. It was a new value, written by a model into the claim file on 10 November 2025, and readable by adjusters, the special investigations unit, the next agent and the guest on request. It inherited the two losses of hops 1 and 2, cited neither field's time basis, and called an arrival a treatment. On 11 November 2025 a claims handler corrected it with the central station record and the PDF's printed "EST". The correction did not overwrite the flag. It was written as a record of its own, with its lineage: the flag it corrects, the records it relied on, the handler who made it, and the time it was made. The next reader of the claim file, person or model, sees the flag, the correction and why. The cost was two hours of a handler's time and the guest's trust (illustrative).

The full Ledger of the night follows: fourteen field lineages, one main line.

THE LEDGER, PART 1 OF 2

One Night, Every Value: the full Ledger: movement and meaning

Hop, year	From	To	Mechanism and schedule	Format change; what was lost or assumed
1 2025	Fire panel 2 event log	Facilities system, then the claim file (CSV export)	The alarm time. The facilities vendor's nightly export sends panel events as text; the claim file imports the export on request	TIME Panel wall time '2025-11-02 01:42:10' as text; no offset, no zone name. **Inside the repeated hour the text names two instants an hour apart; the export says neither which pass nor which zone**
2 2025	Hospital registration (HL7 v2, offset kept; FHIR inside the hospital, offset kept)	PDF records released to the insurer, then its document-extraction service	The arrival time. Released under the guest's written authorisation; the extraction service reads the PDF and fills claim fields	TIME RIGHTS Printed '11/02/2025 01:20 EST' becomes the field value '2025-11-02 01:20'. **The extraction keeps the clock reading and drops 'EST'; the authorisation's description and purpose do not travel with the extracted fields, so later readers cannot tell what the release allowed**
3 2025	EU booking site (UTF-8)	Property system export (Windows-1252 CSV), then the claim file; hospital registration reads the passport's machine readable zone	The guest's name. The property system's legacy export converts text to Windows-1252; the hospital's registration desk scans the passport and takes the name from its machine readable zone	IDENTITY The guest's UTF-8 name (initials I. Y.) loses its three non-ASCII letters (one ş, two ı) in Windows-1252, each replaced by '?'; the passport MRZ gives the name in upper case, surname first, transliterated. **Three spellings of one person; exact and case-insensitive matching both fail; a later UTF-8 file opened as Windows-1252 shows a fourth, mojibake form**
4 2025	EU booking site (DD/MM/YYYY)	Hotel guest profile, then claim intake (MM/DD/YYYY)	The date of birth. Entered as text on the EU form, stored as text in the profile, parsed at claim intake	IDENTITY '03/04/1990' entered as 3 April; parsed as 4 March. **The second matching key fails: the claim intake record and the hospital record now disagree on the date of birth**
5 2025	Insurer's policy system	Broker's spreadsheet, then claim intake	The policy number. The broker cleans a list of policy numbers with find-and-replace before sending it	IDENTITY OWNERSHIP 'CP 0041-7730' loses the prefix, the space and the hyphen; the cell turns '00417730' into the number 417730. **Intake finds no policy 417730; the cleaned sheet has no accountable owner and stays in use for other claims**
6 2025	EU booking engine (EUR)	Group finance (USD), then the payment API (JSON)	The refund. The group refunds the whole three-night booking after the fire; finance converts each night at the last published reference rate and rounds each line half-up; the payments team converts the total once	PRECISION TIME 3 x EUR 411.52 = EUR 1,234.56. Finance converts each night at Friday 31 October's ECB rate (1.1554) and rounds half-up: USD 1,426.41. Payments converts the total at Monday 3 November's rate (1.1514): USD 1,421.47, a cent from the per-line figure at that rate. The two records disagree by USD 4.94. **No reference rate is published on a weekend, so the rate date is an unstated choice (the contract's time-basis line should say: last published rate before the transaction date); the rounding point and rule are unwritten**

Hop, year	From	To	Mechanism and schedule	Format change; what was lost or assumed
7 2025	Laboratory analyser, HL7 v2 OBX (value, unit, reference range)	PDF records, then the claim summary	The carboxyhaemoglobin (COHb) result. The extraction service keeps the result line and drops the notes around it	CONTEXT 'COHb 12 %' kept; the reference range, the specimen time and 'oxygen given before the draw' dropped. **Without the specimen time and the oxygen note, a reader cannot judge the exposure from the number**
8 2025	Hospital coder (ICD-10-CM, edition in force on the date of service)	Itemised bill, then the claim system	The diagnosis code. The bill prints codes; the claim system stores them in a text column	MEANING The code is stored without its code system or edition. **A later reader cannot tell which code system and which edition gave the code its meaning**
9 2025	Property schedule of the policy (latitude, longitude columns)	Claims risk map (GeoJSON, longitude first)	The hotel's location. A script builds GeoJSON points from the schedule's two columns in the order they appear	MEANING (38.9000, -77.0400) written as [38.9000, -77.0400] where GeoJSON expects [longitude, latitude]. **The point lands at latitude -77.04, longitude 38.90, on the Antarctic ice sheet; it passes range checks because both numbers are valid latitudes**
10 2025	Facilities system	Claim file	The inspection date. The export writes an empty string when the field has no value	COMPLETENESS Empty: the export does not say whether the date was never recorded, the inspection was not done, or the field does not apply. **The inspection was done in August 2025 and recorded in the vendor's own system, not in this field; the agent reads the empty field as 'never inspected'**
11 2025	Insurer (pain.001 to its bank)	The bank, then a cross-border pacs.008 to the guest's Turkish IBAN (2 December 2025)	The payment. The insurer's payment file carries the guest's name and address as free-text lines; the bank's own terms require a structured or hybrid address (illustrative)	IDENTITY COMPLETENESS Unstructured address lines mapped into structured fields; town left empty; name limited to the profile's character set (ASCII only for names). **The payment is held for repair at the bank; the name on the payment matches neither the booking nor the passport spelling exactly**
12 2025	Claims database (the claim file's time fields, held as text)	Warehouse files (Parquet)	The claim timestamps. A nightly load converts the claim file's text times and writes them as instants	TIME The alarm text '2025-11-02 01:42:10' is written as a TIMESTAMP with isAdjustedToUTC=true, as if it were UTC. **In the warehouse the alarm now reads 01:42 UTC, which is 21:42 EDT the evening before**
13 2025	Claims notes	Embedding index used by the triage agent to find similar claims	The incident report. After an embedding model upgrade, queries use the new model; the stored vectors were made with the old one	MEANING Same dimension, so nothing fails; the index metadata holds no model identifier. **Similarity between old and new vectors is computed but not meaningful: scores cluster lower and recall drops in evaluation; the fix is model id, dimension and metric in the index metadata, and a refusal on mismatch**

Hop, year	From	To	Mechanism and schedule	Format change; what was lost or assumed
14 2025	Claims-triage agent	Claim file (a new record), 10 November 2025	The flag. The agent compares the two time fields and writes 'medical treatment recorded 22 minutes before the fire alarm'	TIME CONTEXT A sentence and a review status: a new value written by a model. **The flag inherits hops 1 and 2 and cites neither field's time basis; corrected on 11 November 2025 with the station record and the 'EST' on the hospital PDF; its correction and lineage are recorded**

THE LEDGER, PART 2 OF 2

One Night, Every Value: the full Ledger: control, readers and cost

Hop	Accountable owner	Who can change it	Classification; retention or erasure	Read by	Cost per month; basis
1	Hotel facilities manager	Facilities vendor technicians (panel clock and configuration); the export job's service account	Operational safety record; kept for the claim and by fire code	Facilities staff, the fire investigator, the insurer's adjuster and its triage agent	Monthly: negligible to store. One-off: two hours of a handler's time on the wrong flag (hop 14) (not monthly) (illustrative)
2	Hospital health information management (source); insurer claims operations (extracted fields)	Registration clerks (source); the extraction service and claims staff (claim fields)	Protected health information at the hospital, where 45 CFR 164.508 governs the disclosure; at the insurer, state insurance privacy law (many states follow NAIC model regulation 672) governs its use	Clinicians; then the adjuster, the nurse reviewer and the triage agent	Monthly: per-page extraction fees plus human review for each claim file (illustrative)
3	Group guest data team (booking); hospital registration (patient)	Front desk and booking staff; hospital registration clerks	Personal data (GDPR for the EU booking); protected health information at the hospital	Front desk, finance, the claim file, the payment bank	One-off: a manual identity check by the adjuster, about an hour (not monthly) (illustrative)
4	Group guest data team	The guest (on the form); claim intake staff	Personal data	Claim intake, identity matching, the triage agent	One-off: part of the same manual identity check (not monthly) (illustrative)
5	None for the copy (the broker's account team made it); the insurer's policy administration owns the source	Anyone at the broker with the file	Commercial record	Claim intake, the broker	One-off: a two-day delay in opening the claim (not monthly) (illustrative)
6	Group finance	Finance analysts; the payments service	Financial record	Finance, the payment provider, auditors	One-off: a reconciliation break reviewed by finance (not monthly) (illustrative)
7	Hospital laboratory (source); insurer claims operations (summary)	Laboratory staff; the extraction service	Protected health information	Clinicians; the nurse reviewer; the triage agent	One-off: a records request to the hospital, about a week (not monthly) (illustrative)

Hop	Accountable owner	Who can change it	Classification; retention or erasure	Read by	Cost per month; basis
8	Hospital coding (source); insurer claims operations (stored code)	Hospital coders; claims staff	Protected health information	The nurse reviewer, claims analytics, the triage agent	Monthly: small per claim; larger for analytics that join codes across years (illustrative)
9	Insurer underwriting (schedule); catastrophe modelling team (map)	Underwriting assistants; the map script's owner	Commercial record	Catastrophe modellers, the adjuster's map view	One-off: a hotel missing from the accumulation report until someone notices (not monthly) (illustrative)
10	Hotel facilities manager	Facilities staff; the inspection vendor (in its own system)	Operational safety record	The adjuster, the triage agent, the fire investigator	One-off: a coverage question raised and withdrawn (not monthly) (illustrative)
11	Insurer treasury operations	Treasury staff; the bank's repair desk	Financial record; personal data	Banks on the route, sanctions screening, the guest	One-off: a repair fee and a three-day delay (not monthly) (illustrative)
12	Claims data engineering	The load's service account; data engineers	Commercial record	Claims analytics, fraud models, actuaries	Monthly: storage is small; every dashboard built on the field inherits the error (illustrative)
13	Claims AI platform team	Platform engineers	Derived data from personal data	The triage agent	One-off: a re-embedding run for the whole index (not monthly) (illustrative)
14	Claims operations	The triage agent (writes); claims handlers (correct)	Commercial record; may affect a person's claim	Adjusters, the special investigations unit, the guest on request	One-off: two hours of a handler's time and the guest's trust (not monthly) (illustrative)

Every loss in the Ledger happened where a value moved to a reader that lacked, or did not enforce, a place for its rules. After the contracts, the main line reads differently. The panel's export writes the offset and its clock synchronisation is a checked target; the extraction captures the zone label or holds the field; the comparison runs on two instants; and the agent receives both values with their contracts and must ask when a time basis is missing.

After the contracts: the main line again, with the Type Contract at each hop

Hop	From	To	What was lost or assumed
1	Fire panel 2	Claim file	**The export writes RFC 3339 with offset and the zone name; panel clock synchronisation is a checked target; the station record is a cross-check; the time-basis test passes**
2	Hospital PDF	Claim file	**Extraction must capture the zone label or hold the field; authorisation scope travels as a field**
1+2	Two time fields	Comparison	**Both are instants; the comparison is done in UTC; the fall-back-night test case is in the suite**
14	Triage agent	Claim file	**The agent receives both values with their contracts and cites them; it must ask when a time basis is missing**

ERA CARD

Reader. Every reader of a boundary field: the loader, the warehouse, the adjuster, the bank, the agent. Each finds the contract where it already looks.

Security. A held value is a control: a field that fails its contract cannot silently become a decision. The authorisation's scope travelling as a field is a privacy control, not a document.

Cost. Half a lead for a quarter, one engineer for two months and owner hours in the first (illustrative), against corrections that each cost hours and, at scale, trust.

AI. The agent prompt test turns "the model should ask" into a check that runs on every model, prompt or schema change.

Monday rule: add the seven fall-back-night suite cases, including the agent prompt that must return "ambiguous, need offset", to the test suite of every pipeline and agent that reads a time, and fail the build on any silent default.

CLOSING

Epilogue: The Contract Travels or the Meaning Does Not



At 01:42:10 on the night the clocks went back, a fire panel in a hotel kitchen wrote down a time. It was correct, within seconds of the monitoring station's clock. A hospital registration clerk wrote down another time a little later, and it was correct too. Eight days later a model compared the two and wrote that a guest was treated 22 minutes before the fire alarm, when she arrived 38 minutes after it. Nobody in the chain typed a wrong digit. Two temporal types were treated as one, because the rules that told them apart stayed behind at the first hop. That is the whole book in one night: a stored value is usable only when its reader has the rules to interpret it, and the rules travel only if someone makes them travel.

Fourteen lineages, one pattern

The Ledger in Chapter Sixteen lists fourteen field lineages from that night, and they look like fourteen different problems. A name lost three letters to a code page. A date of birth swapped its day and month. A policy number became an integer. A refund depended on a rate date nobody wrote down. A coordinate pair swapped its order and put a hotel on the Antarctic ice sheet. An empty field became "never". A timestamp became UTC because a loader said so. A vector lost the name of the model that made it. Each failure is ordinary, and each has a standard that, chosen and tested at the boundary, would have held the value. What they share is the moment: a value moved to a reader that had no place to find its rules.

Every era kept those rules somewhere: the ruled column heading, the card layout sheet, the copybook, the catalog, the file header, the schema registry, the API document. Every era also had readers that arrived without them. The insurer's own history, told in *From Tally to Token*, is a long record of exactly that. The Type Contract is not a new idea. It is the seven questions every good data steward has always asked, written down in seven lines, attached to the value, and tested where the value crosses.

What the AI reader changes

Earlier readers failed in two ways. Some stopped: a program whose arithmetic met a field that was not valid packed decimal ended with a data exception, and a loader that expected a date and received a name rejected the row. Those failures were expensive, but visible. Others guessed in silence, as most of the Ledger shows: a date parser, a spreadsheet, a map script, a loader that assumed UTC.

A language model changes the failure, not the problem. It reads tokens, and the tokens of "01:42" and "01:20" carry no time basis. When the rules are missing it can produce a fluent, plausible answer instead of stopping, and the answer is written back as a new record that the next reader trusts. The model is also the first machine reader that can be told the rules in plain words with every value and asked to stop when one is missing. That is the opportunity. A compiled program had its record layouts fixed in advance; a model can be given the rules of each value, in words, with the value.

What it does not change

No reader recovers a fact that was never written. The fire panel never recorded an offset, and no model, however capable, can read four tokens that are not there. The central monitoring station's record, read with the "EST" the hospital had printed, settled the night: a third clock had written the instant with its own synchronised time. Somebody had decided, long before the fire, that that clock mattered.

The work is also still human. Somebody owns each code table. Somebody decides the rounding point and the rate date. Somebody decides that absence means "not recorded" and not "never". A model can draft a contract from a schema, and it can notice a gap, but the decisions in the seven lines are commitments a company makes, and they need a name beside them.

> **OUR VIEW, AS OF NOVEMBER 2026**
>
> Models will keep getting better at noticing what is missing, and that is good: an agent that sees 2 November 2025 in America/New_York and asks which 01:42 is a better colleague than one that subtracts. But the companies that get trustworthy AI answers will not be the ones with the best guesser. They will be the ones whose values arrive at the model with their rules attached, whose tool schemas carry units and time bases, whose vector indexes know their own model, and whose test suites include a prompt that must return "ambiguous, need offset". Those controls are cheap to build at the boundary and expensive to reconstruct after a wrong answer has been filed.

> *The representation travels well. The rules travel only if we send them.*

The guest was paid after a three-day repair delay. The flag was corrected, and the correction is a record with its own lineage. The fire panel's next export writes an offset and a zone name. None of that needed a new standard; every standard the fixes rely on existed before the night began. It needed a decision, field by field, that the meaning of a value is part of the value. The contract travels, or the meaning does not.

APPENDICES

APPENDIX A

The Type Contract: Template, Tests and Mappings

The Type Contract as a working tool: the blank template with a test and an owner for each line, the alarm time of the night filled in after the fix, and a map of each line to the places readers already look: SQL DDL, Parquet, Avro, JSON Schema and FHIR. The blank template and this worked example are a free download on the book's web page, as an XLSX workbook and a PDF (blog.jcardena.com/en/ebooks/what-a-value-means/).

The blank template

Write one contract per field that crosses a boundary, not per column in the estate. Each line names a rule and the test that holds a value failing it at the boundary; "not applicable" is a valid answer and is itself tested (an unexpected unit on a count is an error).

Line	What it says	Test at the boundary	Who owns it
1. Type, precision and rounding	Logical type; precision and scale; the rounding rule and the point where rounding happens; the allowed range	Value parses as the type; digits within precision and scale; inside the range; a known tie case rounds as written	The field's data owner
2. Encoding and serialization	Character set, byte order, the text form of the value	Bytes decode in the named encoding without replacement; the text matches the named form	The producing system's engineering lead
3. Unit or currency	A UCUM unit or an ISO 4217 code, in its own field or fixed by the contract	The code is in the named code list; the unit fits the quantity	The business owner of the quantity (finance for money)
4. Time basis	Instant, wall time, date or duration; the IANA zone name; the clock source and how it is synchronised; for rates, which date's rate	Instants carry an offset; the offset is valid for the zone at that instant; a fall-back-night case is in the suite	The producing system's owner, with the clock's owner
5. Identity scope	Which entity the value belongs to, and which key system names it	The key is present, well formed (check digit where one exists) and resolves in the named system	The owner of the key system
6. Absence	What missing, null, empty and zero each mean, and which are allowed	Only the allowed kinds of empty pass; each carries its reason	The field's data owner
7. Authority	Code system and version; the accountable owner; which source wins on conflict; who may receive the value	The code exists in that version; the recipient is on the list; a conflict holds the value	The code table's owner, named by job title

A value that fails any line is held with its reason, not passed with a warning; a reader may see it only with that reason stated. The owner column names a role, not a team inbox; a line with no owner is a finding.

The worked example: the alarm time after the fix

Hop 1 of *One Night, Every Value* (illustrative) carried the panel's wall time "2025-11-02 01:42:10" with no offset. After the fix the export writes RFC 3339 text with the offset and, per RFC 9557, the zone name in brackets: `2025-11-02T01:42:10-04:00[America/New_York]`. The offset says which of the two 01:42:10s it was.

TYPE CONTRACT: ALARM TIME, FIRE PANEL 2

1. Type, precision and rounding. An instant, to the second; no rounding. *Test:* parses as an instant with seconds present.

2. Encoding and serialization. RFC 3339 date-time text in UTF-8, with the RFC 9557 zone suffix. *Test:* the date-time before the bracket matches the RFC 3339 grammar, in which the offset is mandatory, and the whole value matches RFC 9557; text without an offset is held.

3. Unit or currency. Not applicable. *Test:* no unit field is present.

4. Time basis. Instant with offset; zone America/New_York; panel clock synchronised by NTP, a checked target. *Test:* the offset is valid for the zone at that moment (on 2 November 2025 both minus 04:00 and minus 05:00 are valid for 01:42:10, and the offset decides); the panel time agrees with the central station record, 05:42:13 UTC, within the tolerance the owner sets.

5. Identity scope. The alarm event of panel 2 at the hotel. *Test:* panel and event identifiers present and known to the facilities system.

6. Absence. A missing time carries a coded reason (panel offline, event not exported, export failed, unknown); never an empty string. *Test:* an empty string is held; a missing value without a reason code is held.

7. Authority. Owner: the hotel facilities manager. The central station record wins on conflict. Received by facilities staff, the fire investigator, the insurer's adjuster and its triage agent. *Test:* a conflict with the station record holds the field for review.

Where each line goes

No single format carries all seven lines. The table shows where each line fits in five common places, by their own specifications; where a format has no slot, the line must travel as a comment, a documentation string or custom metadata that the format carries but does not check.

Line	SQL DDL	Parquet	Avro	JSON Schema	FHIR (R5)
1. Type, precision, rounding	`NUMERIC(p,s)`, `CHECK` for range; the tie rule is the engine's	`DECIMAL` logical type with precision and scale	`decimal` logical type with precision and scale	`type`, `multipleOf`, `minimum`, `maximum`	`decimal`, whose written precision is significant
2. Encoding, serialization	Column type; the export's text form is outside the DDL	`STRING` is UTF-8	Strings are UTF-8 by the spec	`format` names a text form; JSON between systems is UTF-8	Fixed by the JSON and XML formats
3. Unit, currency	A code column with `CHECK`, or a comment	Key-value metadata	Custom attribute	A sibling field with `enum`; `description`	`Quantity` system and code (UCUM); `Money.currency` (ISO 4217)
4. Time basis	`TIMESTAMP WITH TIME ZONE`; zone name in its own column	`TIMESTAMP` with `isAdjustedToUTC`	`timestamp-*` or `local-timestamp-*`	`format: date-time` (offset required by RFC 3339; an annotation unless asserted, so add a `pattern`)	`instant` requires an offset; `dateTime` requires one when hours and minutes are given; zone name by the `tz-code` extension
5. Identity scope	`PRIMARY KEY`, `FOREIGN KEY`, `UNIQUE`	`UUID` logical type; scope in key-value metadata	`uuid` logical type; scope in `doc`	`pattern`; scope in `description`	`Identifier.system` and `value`
6. Absence	`NOT NULL`; one kind of null	Optional or required; one kind of null	Union with `"null"`; one kind	`required` (absent) apart from `null`	`data-absent-reason` extension with coded reasons
7. Authority	`REFERENCES` a code table; `GRANT` for readers	Key-value metadata	`enum` symbols; custom attribute	`enum`; no version slot	Binding to a value set with a strength; `Coding.system` and `version`; `Meta.security`

What each cannot carry

SQL DDL. The standard gives types and constraints; documentation is vendor syntax. PostgreSQL's manual says "There is no COMMENT command in the SQL standard", and that its `numeric` type "rounds ties away from zero": the rounding rule belongs to the engine, not the column. A PostgreSQL `timestamp with time zone` is stored as UTC and "the originally stated or assumed time zone is not retained", so the zone name needs its own column.

Parquet. Logical types annotate physical ones: `DECIMAL` over INT32, INT64 or byte arrays, and `TIMESTAMP` with a unit and the `isAdjustedToUTC` flag. When the flag is true the value is an instant and, in the specification's words, "time zone information gets lost". Units, currencies, owners and code versions go only in `key_value_metadata`, string pairs at file or column level that no reader is required to check.

Avro. The specification permits attributes it does not define "as metadata", so a `unit` or `owner` attribute travels, unchecked. An invalid logical type is ignored and the reader falls back to the underlying type: a `decimal` whose scale exceeds its precision is read as its underlying `bytes` or `fixed` value.

JSON Schema. In draft 2020-12, `format` is an annotation unless the validator turns assertion on, so `date-time` is advice by default. JSON numbers have no declared precision, and RFC 8259 notes that interoperability is good only within binary64; send amounts as strings with a pattern, or as integers of the minor unit. `enum` lists codes but not the code system's version.

FHIR. FHIR carries the most: an offset on every `instant`, the code system and its version on a `Coding`, coded reasons for absence, and security labels. It cannot carry rounding rules, and its `Meta.security` labels are, by the specification, for finding and controlling access to a resource, not for interpreting its content. A rule beyond the base specification travels as an extension, which every element may hold and every extension names by URL.

Monday rule: for each boundary field, fill the seven lines in the template, then copy each line into the slot this table names for your format, and put the rest in metadata with a test that reads it.

APPENDIX B

Format Decision Table

Five jobs, the formats commonly chosen for each, and what each format keeps and loses of a value's rules, by its own specification. No format keeps all seven lines of the Type Contract; the right question is which lines the job needs and where the rest will travel (Appendix A).

Job	Format	Keeps	Loses
Interchange between companies	Industry message (FHIR R5)	An offset on every `instant`; code system and version on a `Coding`; UCUM units on a `Quantity`; coded reasons for absence	Rounding rules; any rule beyond the base specification travels as an extension
	JSON (RFC 8259) with JSON Schema	Names, nesting, strings, numbers, booleans, null; UTF-8 between systems	No date type; numbers interoperate only within binary64; `format` is advice unless asserted; duplicate names give unpredictable results
	XML with XML Schema 1.1	`xs:decimal`, dates and times; `xs:dateTimeStamp` requires an offset	Plain `xs:dateTime` lets the offset be left out
	CSV (RFC 4180)	Rows and fields as text	Every type; the header is optional; the character set travels only in a MIME parameter that a file on disk does not have
Internal events	Avro with its schema	Logical types (`decimal`, `uuid`, instant and local timestamps); `doc` strings; resolution between writer and reader schemas	The zone name; custom attributes travel unchecked; an invalid logical type falls back to the underlying type
	Protocol Buffers (proto3)	Field numbers that never change; unknown fields preserved on parse and reserialisation	No decimal type; a singular scalar field without `optional` cannot tell zero from not set
Analytics storage	Parquet	`DECIMAL` precision and scale; `TIMESTAMP` unit and `isAdjustedToUTC`; `STRING` as UTF-8; key-value metadata	The zone name; units and code versions except as unchecked metadata; one kind of null. INT96 is deprecated for new writers
	Arrow (in memory and IPC)	A timestamp's tz database zone name or offset; decimals of 32 to 256 bits; custom metadata on fields and schemas	Rounding rules, units and code versions except as metadata
Archive	PDF/A (ISO 19005)	The page as a person reads it; PDF/A-3 (ISO 19005-3:2012) can embed files of any type alongside	Fields become printed text; a reader must extract them again, which is where hop 2 lost "EST"
Model input	JSON plus a tool or output schema (JSON Schema)	Field names, types, `enum` lists and `description` strings, all placed in the model's context as text	The model reads tokens, not types; a vendor's strict mode can constrain the shape of what it writes, but nothing checks a value's truth unless the caller does

How to read the table

Interchange. Prefer the industry message where one exists: it fixes more lines than a general format, though agreements between the parties still own the rest (Chapter Thirteen). Where none exists, JSON with a published schema is the floor; send amounts as strings or as integers of the minor unit, and times as RFC 3339 text.

Events and storage. Avro, Protocol Buffers and Parquet carry types well and meaning poorly. Their metadata slots can hold the remaining lines, but only a test that reads the metadata makes them a contract (Chapters Twelve and Fourteen).

Archive. PDF/A keeps what a person saw, not what a program needs. When the record will be read by software later, keep the structured value beside the page, with its Type Contract.

Model input. The schema reaches the model as text, so its `description` strings are where the time basis, the unit and the meaning of empty must be written (Chapter Fifteen).

APPENDIX C

Failure Library

The sourced failure cases in this book, each with the official report or paper behind it and the line of the Type Contract it broke. The line is the author's reading of the case, not a finding of the report; every fact in the table is from the source named. Each case is told once, in the chapter given, and two cases told in *From Tally to Token* appear here only as pointers.

Case, year	What happened, by the source	Line broken	Official report or paper	Ch
Gimli Glider, 1983	On 23 July an Air Canada Boeing 767, C-GAUN, made an emergency landing at Gimli, Manitoba, after it "ran out of fuel over Red Lake, Ontario" (p. 9). The 767 measured fuel in kilograms; the crew and ground staff in Montreal were given the conversion factor 1.77, which converts litres to pounds. 7,682 litres times 1.77 gave 13,597, "in pounds but everyone involved believed it to be kilograms" (p. 41); the correct load was about four times what was added (p. 42). The Board called the "metric mix-up" explanation "simplistic and inaccurate" (p. 35): inoperative fuel gauges, the minimum equipment list and training also contributed.	3. Unit (with 7: the Board found a failure to assign clearly the responsibility for the manual fuel calculation, p. 36)	*Final Report of the Board of Inquiry Investigating the Circumstances of an Accident Involving the Air Canada Boeing 767 Aircraft C-GAUN that Effected an Emergency Landing at Gimli, Manitoba on the 23rd Day of July, 1983*, Commissioner the Hon. Mr. Justice George H. Lockwood, April 1985, Catalogue No. T22-64/1985E. Copy hosted by the FAA, faa.gov	8
Patriot at Dhahran, 1991	On 25 February 1991 a Patriot battery failed to track an incoming Scud, which hit an Army barracks, killing 28 Americans (p. 1). Time was kept in tenths of seconds as an integer, and because the registers "are only 24 bits long" its conversion to a real number could be no more precise than 24 bits (p. 5). After about 20 hours of continuous use the target fell outside the range gate; after 100 hours the time was off by .3433 seconds, a shift of about 687 metres (p. 15). The battery had run for about 100 hours.	1. Type, precision and rounding (with 4: the error grew with time since start)	US General Accounting Office, *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia*, GAO/IMTEC-92-26, B-247094, 4 February 1992, released 27 February 1992. gao.gov/products/imtec-92-26	3

Case, year	What happened, by the source	Line broken	Official report or paper	Ch
Boeing 787 generator control units, 2015	The FAA found that a 787 "powered continuously for 248 days can lose all alternating current (AC) electrical power" because a software counter inside the generator control units "will overflow after 248 days of continuous power". The interim fix: switch the electrical power off at intervals not to exceed 120 days. AD 2018-20-15 later required new software.	1. Type, precision and rounding (the counter's range)	FAA, Airworthiness Directive 2015-09-07, Amendment 39-18153, Docket FAA-2015-0936, 80 FR 24789, effective 1 May 2015; superseded by AD 2018-20-15, 83 FR 52131. govinfo.gov	3
Gene symbols in spreadsheets, 2016	Microsoft Excel, "when used with default settings", converts gene names to dates and floating-point numbers: SEPT2 becomes "2-Sep", MARCH1 "1-Mar", and, as an earlier report they cite had shown, the identifier 2310009E13 becomes "2.31E+13". The authors screened 35,175 supplementary Excel files from 18 journals, 2005 to 2015, and confirmed errors in files from 704 of 3,597 papers with gene lists: 19.6 per cent. In 2019 the HGNC renamed such symbols, for example SEPT1 to SEPTIN1 and MARCH1 to MARCHF1, as its 2020 guidelines explain.	1. Type (identifiers read as dates and numbers), with 5. Identity scope	Mark Ziemann, Yotam Eren and Assam El-Osta, "Gene name errors are widespread in the scientific literature", *Genome Biology* 17, 177, 23 August 2016. doi.org/10.1186/s13059-016-1044-7; Elspeth A. Bruford et al., "Guidelines for human gene nomenclature", *Nature Genetics* 52, 754 to 758, 2020	7

Case, year	What happened, by the source	Line broken	Official report or paper	Ch
Ariane 501, 1996	Told in *From Tally to Token*. The Inquiry Board found the inertial reference software raised an exception converting a 64-bit floating-point value to a 16-bit signed integer, the value being too large to represent (p. 4).	1. Type, precision and rounding (range)	*Ariane 5 Flight 501 Failure: Report by the Inquiry Board*, chaired by Prof. J. L. Lions, Paris, 19 July 1996. esamultimedia.esa.int/docs/esa-x-1819eng.pdf	Book 1
Mars Climate Orbiter, 1999	Told in *From Tally to Token*. The Board found the root cause was "the failure to use metric units in the coding of a ground software file, 'Small Forces', used in trajectory models" (p. 16).	3. Unit	*Mars Climate Orbiter Mishap Investigation Board Phase I Report*, NASA, 10 November 1999. llis.nasa.gov	Book 1

How to use the library

Read the cases by line, not by industry. Four of the six broke line 1: a value outgrew the range or precision its type allowed (Patriot, the 787 counter, Ariane 501), or a reader chose a type the writer never meant (the gene symbols). A range test at the boundary, and a test for how long the value may accumulate, is cheap compared with any of them. The two unit cases (Gimli and Mars Climate Orbiter) are the reason line 3 asks for the unit in its own field, from a named code system, and not in a column heading or a habit.

None of these cases had a single cause, and the reports say so; the Gimli Board was explicit. The Type Contract does not replace an investigation. It names the rule that, written down and tested, would have held the value at the boundary before a person or a model acted on it.

APPENDIX D

Glossary

Terms used in this book, in plain words, as of November 2026.

Term	Meaning
Absence	Line 6 of the Type Contract: what missing, null, empty and zero each mean.
Axis order	Which coordinate comes first. GeoJSON puts longitude first; the EPSG definition of code 4326 puts latitude first.
Binary64	The IEEE 754 64-bit binary floating-point format; it holds most decimal fractions only approximately.
Check digit	A digit computed from the others in a key, so that most typing errors fail a test.
Code point	A number Unicode assigns to a character, from 0 to hexadecimal 10FFFF.
Code system	A versioned published list that gives codes their meaning, such as ICD-10-CM.
Confusable	A character that looks like another; an attack surface in names and identifiers.
CRS	Coordinate reference system: the datum, axes and units that make two numbers a place.
Decimal type	A type that stores base-ten digits exactly, with a declared precision and scale.
Embedding	A vector a model computes from text; meaningful only with the model that made it.
Encoding	The rule that turns code points into bytes, such as UTF-8 or Windows-1252.
Enumeration	A field limited to a fixed list of values, each with a stated meaning.
Epoch	The starting instant a count of time is measured from, such as 1 January 1970 UTC.
Fall-back night	The night clocks go back an hour, so one hour of wall time happens twice.
Half-even	Rounding that sends an exact tie to the even digit: 2.5 to 2, 3.5 to 4.
Half-up	Here, a tie goes away from zero: 2.5 to 3, minus 2.5 to minus 3; some systems give minus 2.
Hop	A move or change after which a different reader holds the value.
IANA time zone database	The public record of each zone's offset and daylight saving rules over time.
Instant	A single point on the time-line, the same everywhere; written in UTC or with an offset.
isAdjustedToUTC	The Parquet timestamp flag: true means the value is an instant, false a local wall time.

Term	Meaning
Leap second	A second added to (or, in principle, removed from) UTC to keep it near Earth's rotation; under CGPM 2026 Resolution 5, continuous UTC begins on 20 May 2027.
Logical type	A meaning laid over a physical type, such as a date stored as a 32-bit integer.
Loss code	One of the eight Ledger labels for what a hop lost, such as TIME or IDENTITY.
Lossy transcoding	Converting text to an encoding that lacks some characters; they become "?".
Minor unit	The number of decimal places ISO 4217 gives a currency: 2 for the euro, 0 for the yen.
Mojibake	Garbled text from reading bytes in the wrong encoding; the bytes are intact.
NaN	"Not a number": an IEEE 754 floating-point value for an undefined result, such as zero divided by zero.
Natural key	An identifier taken from the world, such as a policy number.
Normalization	Turning text into one standard form (UAX #15), so the same characters compare equal.
Null	A marker that a value is absent; it does not say why.
Offset	The difference from UTC at an instant, such as minus 04:00; not a zone.
Physical type	How a format stores bits, such as a 64-bit integer, before any meaning is added.
Precision	The total number of significant digits a value may hold.
Reference rate	A published exchange rate for a date, from a named source.
Scale	The number of digits after the decimal point.
Schema registry	A service that stores schema versions and checks changes against a compatibility rule.
Schema resolution	Avro's rules for reading data written with one schema using another.
Surrogate key	An identifier a system invents, with no meaning outside it.
Time basis	Line 4 of the Type Contract: instant, wall time, date or duration, with zone and clock source.
Tokenizer	The program that splits text into a model's tokens.
Type Contract	Seven tested lines that state the rules for reading a field (Appendix A).
UCUM	The Unified Code for Units of Measure: codes for units in electronic messages, such as mg/dL.
UTC	Coordinated Universal Time, the reference for offsets and instants.
UTF-8	The Unicode encoding that uses one to four bytes per code point.
Value	A representation together with the rules that let a reader interpret it.
Vector	A fixed-length list of numbers; it needs dimension, element type, metric and model id.

Term	Meaning
Wall time	A clock reading without an offset; ambiguous in a repeated hour.
Zone	A named region with one history of offsets, such as America/New_York.

APPENDIX E

Sources

Every dated fact in this book was checked against the source listed for its chapter in November 2026 (edition 1.1 rechecks on 20 November 2026). Primary sources come first: original papers, standards, statutes, judgments, official reports, vendor announcements and institutional histories. This list holds 360 sources; the claim-by-claim record behind it is kept with the book and is available on request.

Reader's Brief: One Night, Seven Lines

- The book's running example, One Night, Every Value (illustrative
- 15 U.S.C. 260a, LII https://www.law.cornell.edu/uscode/text/15/260a

The Patient Who Arrived Before the Fire

- The book's running example, One Night, Every Value (illustrative
- 15 U.S.C. 260a, LII https://www.law.cornell.edu/uscode/text/15/260a
- Computed for this book with Python 3.13, zoneinfo and the IANA time zone database
- 45 CFR 164.508(c)(1), "Uses and disclosures for which an authorization is required", LII https://www.law.cornell.edu/cfr/text/45/164.508
- 45 CFR 164.508(c)(1)(iv), eCFR https://www.ecfr.gov/current/title-45/subtitle-A/subchapter-C/part-164/subpart-E/section-164.508

Chapter 1. What a Value Is

- William Kent, Data and Reality, author's page https://bkent.net/Doc/darxrp.htm
- ISO/IEC 11179-1:2023, Information technology, Metadata registries (MDR), Part 1: Framework, foreword and introduction (publisher preview) https://www.iso.org/standard/78914.html
- Apache Parquet, Logical Types https://parquet.apache.org/docs/file-format/types/logicaltypes/
- Apache Avro 1.12.0 Specification https://avro.apache.org/docs/1.12.0/specification/
- RFC 3339, Date and Time on the Internet: Timestamps, section 5.6 https://www.rfc-editor.org/rfc/rfc3339
- 15 U.S.C. 260a, LII https://www.law.cornell.edu/uscode/text/15/260a
- The book's running example, One Night, Every Value (illustrative

Chapter 2. Where the Contract Lives

- IBM, Form and Card Design, C20-8078 (1961), PDF pages 13 and 17 https://ftp.mirrorservice.org/sites/www.bitsavers.org/pdf/ibm/generalInfo/C20-8078_Form_and_Card_Design_1961.pdf
- Federal Register, Vol. 62, No. 145, 29 July 1997, NIST notice of withdrawal of FIPS https://www.govinfo.gov/content/pkg/FR-1997-07-29/html/97-19839.htm
- Computerworld, 18 November 1985, letters, Google Books https://books.google.com/books?id=O90MzD9_kawC; Computerworld, 13 January 1986, In Depth 1985 index, Google Books https://books.google.com/books?id=TC15Q4pvUGIC
- E. F. Codd, The Relational Model for Database Management, Version 2 (Addison-Wesley, 1990), Google Books https://books.google.com/books?id=ZZckAQAAIAAJ
- W3C, XML Schema Part 2: Datatypes, sections 3.2.3 and 3.2.7 https://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
- Confluent documentation, Schema Registry overview https://docs.confluent.io/platform/current/schema-registry/index.html
- Apache Avro 1.12.0 Specification https://avro.apache.org/docs/1.12.0/specification/

- OpenAPI Specification v3.0.0 https://spec.openapis.org/oas/v3.0.0.html
- RFC 3339, sections 4.3 and 5.6 https://www.rfc-editor.org/rfc/rfc3339
- ISO/IEC 11179-1:2023, introduction (publisher preview) https://www.iso.org/standard/78914.html
- Computerworld, "Codd's 12 rules", 2 September 2002 https://www.computerworld.com/article/1381158/codd-s-12-rules.html
- Confluent documentation, Formats, Serializers and Deserializers https://docs.confluent.io/platform/current/schema-registry/fundamentals/serdes-develop/index.html
- W3C, XML Schema Part 2: Datatypes, section 3.2.7.4 https://www.w3.org/TR/xmlschema-2/

Chapter 3. Integers and Floats

- Federal Register 80 FR 24789 (1 May 2015), FR Doc. 2015-10066 https://www.govinfo.gov/content/pkg/FR-2015-05-01/html/2015-10066.htm
- same document, Summary and Discussion
- same document, Discussion and paragraph (g
- Federal Register 83 FR 52131 (16 October 2018) https://www.govinfo.gov/content/pkg/FR-2018-10-16/html/2018-22152.htm
- Java Language Specification SE 21, section 4.2.1 https://docs.oracle.com/javase/specs/jls/se21/html/jls-4.html
- same, section 4.2.2
- same, section 4.2
- PostgreSQL documentation, 8.1 Numeric Types https://www.postgresql.org/docs/current/datatype-numeric.html
- same, 8.1.2
- Apache Parquet format, LogicalTypes.md, INT https://github.com/apache/parquet-format/blob/master/LogicalTypes.md
- IEEE Standards Association https://standards.ieee.org/ieee/754/993/
- IEEE Standards Association https://standards.ieee.org/ieee/754/4211/
- IEEE Microprocessor Standards Committee, ANSI/IEEE Std 754-2019 https://grouper.ieee.org/groups/msc/ANSI_IEEE-Std-754-2019/
- IEEE Standards Association https://standards.ieee.org/ieee/754/6210/
- ISO/IEC JTC 1/SC 22/WG 14, N2531 "C support for ISO/IEC 60559:2020", 15 May 2020 https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2531.pdf
- ECMA-262 current draft, 6.1.6.1 The Number Type https://tc39.es/ecma262/
- same, 21.1.2.6
- same, 6.1.6.2 The BigInt Type
- same, SerializeJSONProperty
- Ecma International, ECMA-262 11th edition, Introduction https://262.ecma-international.org/11.0/
- IETF RFC 8259, section 6 https://www.rfc-editor.org/rfc/rfc8259.html
- U.S. General Accounting Office https://www.gao.gov/products/imtec-92-26; (PDF https://www.gao.gov/assets/imtec-92-26.pdf cover and p. 1
- same report, p. 1 and p. 9
- same report, p. 4
- same report, pp. 5 to 6
- same report, pp. 7 to 9
- same report, pp. 1, 9
- same report, Appendix II, p. 15
- Author's computation, Python 3.13.3 fractions, 20 November 2026
- Author's computation, Python 3.13.3 (math.nextafter, math.ulp, decimal), 20 November 2026
- Author's computation, Node.js 22.20.0, 20 November 2026
- Java Language Specification SE 21, section 15.18.2 https://docs.oracle.com/javase/specs/jls/se21/html/jls-15.html
- M. Cowlishaw, General Decimal Arithmetic https://speleotrove.com/decimal/
- ISO, ISO/IEC/IEEE 60559:2011 https://www.iso.org/standard/57469.html; IEC Webstore https://webstore.iec.ch/publication/11979
- Author's computation, Python 3.13 fractions, 20 November 2026

Chapter 4. Money and Decimals

- PostgreSQL documentation, 8.1.2 Arbitrary Precision Numbers https://www.postgresql.org/docs/current/datatype-numeric.html
- same, 8.1.2
- Microsoft Learn, decimal and numeric (Transact-SQL) https://learn.microsoft.com/en-us/sql/t-sql/data-types/decimal-and-numeric-transact-sql
- IBM Db2 11.5 documentation, Numbers https://www.ibm.com/docs/en/db2/11.5?topic=list-numbers
- Apache Parquet format, LogicalTypes.md, DECIMAL https://github.com/apache/parquet-format/blob/master/LogicalTypes.md
- M. Cowlishaw, General Decimal Arithmetic https://speleotrove.com/decimal/
- IEEE Standards Association, IEEE 754-2019 https://standards.ieee.org/ieee/754/6210/
- Python 3 documentation, decimal module https://docs.python.org/3/library/decimal.html
- Java SE 21 API, java.math.RoundingMode https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/math/RoundingMode.html
- SIX, ISO 4217 Currency Code List One (XML) https://www.six-group.com/dam/download/financial-information/data-center/iso-currrency/lists/list-one.xml; SIX data standards page https://www.six-group.com/en/products-services/financial-information/data-standards.html
- same list, parsed by the author, 20 November 2026
- Publications Office of the EU, CELEX 31997R1103 http://publications.europa.eu/resource/celex/31997R1103; EUR-Lex https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:31997R1103
- same regulation, Article 4
- same regulation, Article 5
- European Central Bank, Euro foreign exchange reference rates https://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates/html/index.en.html
- ECB Data Portal, series EXR.D.USD.EUR.SP00.A https://data.ecb.europa.eu/data/datasets/EXR/EXR.D.USD.EUR.SP00.A; (API https://data-api.ecb.europa.eu/service/data/EXR/D.USD.EUR.SP00.A
- Author's computation, Python 3.13.3 decimal (ROUND_HALF_UP), 20 November 2026
- Author's computation, Python 3.13.3, 20 November 2026
- ECB Data Portal API, series EXR.D..EUR.SP00.A and EXR.D.BGN.EUR.SP00.A https://data-api.ecb.europa.eu/service/data/EXR/D..EUR.SP00.A
- Author's computation, Python 3.13 decimal and float, 20 November 2026
- Microsoft Learn, decimal and numeric (Transact-SQL); PostgreSQL documentation 8.1.2

Chapter 5. Text

- Unicode Consortium, Unicode 18.0.0 https://www.unicode.org/versions/Unicode18.0.0/; Enumerated Versions, publication dates https://www.unicode.org/history/publicationdates.html
- The Unicode Standard, Version 18.0, Core Specification, chapter 2 https://www.unicode.org/versions/Unicode18.0.0/core-spec/chapter-2/
- The Unicode Standard, Version 18.0, Core Specification, chapter 3, sections 3.9 and 3.10 https://www.unicode.org/versions/Unicode18.0.0/core-spec/chapter-3/
- The Unicode Standard, Version 15.0, chapter 2 (PDF) https://www.unicode.org/versions/Unicode15.0.0/ch02.pdf
- The Unicode Standard, Version 18.0, chapter 23 https://www.unicode.org/versions/Unicode18.0.0/core-spec/chapter-23/
- Unicode Consortium, FAQ: UTF-8, UTF-16, UTF-32 and BOM https://www.unicode.org/faq/utf_bom.html
- Microsoft Learn, Code Page Identifiers https://learn.microsoft.com/en-us/windows/win32/intl/code-page-identifiers
- Unicode Consortium, vendor mappings, CP1252.TXT (Microsoft, table version 2.01) https://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP1252.TXT
- Unicode Consortium, vendor mappings, CP1254.TXT https://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP1254.TXT
- Unicode Consortium, vendor mappings, bestfit1252.txt https://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WindowsBestFit/bestfit1252.txt
- Microsoft Learn, WideCharToMultiByte function https://learn.microsoft.com/en-us/windows/win32/api/stringapiset/nf-stringapiset-widechartomultibyte
- Microsoft Learn, char and varchar (Transact-SQL) https://learn.microsoft.com/en-us/sql/t-sql/data-types/char-and-varchar-transact-sql

- Author's computation, Python 3.13.3 standard library (codecs, unicodedata), checked against the Microsoft tables above
- Unicode Standard Annex #15 https://www.unicode.org/reports/tr15/
- Unicode Character Database, SpecialCasing.txt https://www.unicode.org/Public/UCD/latest/ucd/SpecialCasing.txt
- Unicode Technical Standard #10 https://www.unicode.org/reports/tr10/
- Unicode Technical Standard #39 https://www.unicode.org/reports/tr39/
- W3C Internationalization, Personal names around the world https://www.w3.org/International/questions/qa-personal-names
- ICAO, Doc 9303 Machine Readable Travel Documents, Part 3 https://www.icao.int/sites/default/files/publications/DocSeries/9303_p3_cons_en.pdf
- ICAO, Doc 9303 Part 4 https://www.icao.int/sites/default/files/publications/DocSeries/9303_p4_cons_en.pdf
- Unicode Consortium vendor mappings, bestfit1252.txt and CP1252.TXT https://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/
- UAX #29 https://www.unicode.org/reports/tr29/
- Microsoft Learn, Double-Byte Character Sets https://learn.microsoft.com/en-us/windows/win32/intl/double-byte-character-sets

Chapter 6. Time

- Computed for this book with Python 3.13.3, zoneinfo and tzdata 2025b
- Python documentation, datetime https://docs.python.org/3/library/datetime.html
- Python documentation, zoneinfo https://docs.python.org/3/library/zoneinfo.html
- 15 U.S.C. 260a, LII https://www.law.cornell.edu/uscode/text/15/260a
- House Clerk, Roll Call 238 (2026) https://clerk.house.gov/Votes/2026238
- Congress.gov bill actions (API) https://www.congress.gov/bill/119th-congress/house-bill/139/all-actions; H.R. 139 RFS, GovInfo https://www.govinfo.gov/content/pkg/BILLS-119hr139rfs/html/BILLS-119hr139rfs.htm
- EUR-Lex, Directive 2000/84/EC https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32000L0084
- IANA, Time Zones https://www.iana.org/time-zones
- IANA tz NEWS https://data.iana.org/time-zones/tzdb/NEWS
- RFC 6557 https://www.rfc-editor.org/rfc/rfc6557
- BIPM, Resolution 4 of the 27th CGPM (2022) https://www.bipm.org/en/committees/cg/cgpm/27-2022/resolution-4 doi 10.59161/CGPM2022RES4E
- BIPM, 27th meeting of the CGPM https://www.bipm.org/en/committees/cg/cgpm/27-2022; 28th meeting of the CGPM https://www.bipm.org/en/committees/cg/cgpm/28-2026
- UT1-UTC
- IERS, Leap_Second.dat https://hpiers.obspm.fr/iers/bul/bulc/Leap_Second.dat
- IERS, Bulletin C https://hpiers.obspm.fr/iers/bul/bulc/bulletinc.dat
- The Open Group Base Specifications Issue 8 (IEEE Std 1003.1-2024), XBD 3.125 and 4.19 https://pubs.opengroup.org/onlinepubs/9799919799/basedefs/V1_chap03.html; https://pubs.opengroup.org/onlinepubs/9799919799/basedefs/V1_chap04.html
- The Open Group Base Specifications Issue 8, sys/types.h https://pubs.opengroup.org/onlinepubs/9799919799/basedefs/sys_types.h.html
- ISO https://www.iso.org/standard/70907.html
- RFC 3339 https://www.rfc-editor.org/rfc/rfc3339
- RFC 9557 https://www.rfc-editor.org/rfc/rfc9557
- RFC 5905 https://www.rfc-editor.org/rfc/rfc5905
- Microsoft Support, Change the date system, format, or two-digit year interpretation https://support.microsoft.com/en-us/excel/change-the-date-system-format-or-two-digit-year-interpretation
- Microsoft Learn, Excel incorrectly assumes that the year 1900 is a leap year https://learn.microsoft.com/en-us/troubleshoot/microsoft-365-apps/excel/wrongly-assumes-1900-is-leap-year
- eCFR, 45 CFR 164.508(c) https://www.ecfr.gov/current/title-45/subtitle-A/subchapter-C/part-164/subpart-E/section-164.508
- NAIC Model Regulation 672 https://content.naic.org/sites/default/files/model-law-672.pdf
- Computed for this book with Python 3.13 zoneinfo and the IANA data
- IANA tz, Theory and pragmatics of the tz code and data https://data.iana.org/time-zones/theory.html

Chapter 7. Identity: Keys and Identifiers

- Google Patents, US2950048A https://patents.google.com/patent/US2950048A/en
- ISO, ISO/IEC 7812-1:2017 https://www.iso.org/standard/70484.html; preview of the standard (title page, foreword, contents) https://assets.vde-verlag.de/iec-normen/preview-pdf/info_isoiec7812-1%7Bed5.0%7Den.pdf
- CMS, Requirements for National Provider Identifier (NPI) and NPI Check Digit (23 January 2004) https://www.cms.gov/Regulations-and-Guidance/Administrative-Simplification/NationalProvIdentStand/Downloads/NPIcheckdigit.pdf
- ISO, ISO 13616-1:2020 https://www.iso.org/standard/81090.html
- ISO 13616:1997 sample https://cdn.standards.iteh.ai/samples/21228/8c19005e318742e7ba49cd4fe544257c/ISO-13616-1997.pdf
- Computed for this book with Python 3.13, following the ISO 13616:1997 Annex B procedure and the CMS statement of the Luhn formula
- IETF, RFC 9562 https://www.rfc-editor.org/rfc/rfc9562.html
- ISO, ISO 8601-1:2019 https://www.iso.org/standard/70907.html; ISO, ISO 8601 date and time format https://www.iso.org/iso-8601-date-and-time-format.html
- Genome Biology (PMC4994289) https://pmc.ncbi.nlm.nih.gov/articles/PMC4994289/ doi 10.1186/s13059-016-1044-7
- Nature Genetics author manuscript (PMC7494048) https://pmc.ncbi.nlm.nih.gov/articles/PMC7494048/ doi 10.1038/s41588-020-0669-3

Chapter 8. Codes, Units and Enumerations

- CDC, National Center for Health Statistics, ICD-10-CM files https://www.cdc.gov/nchs/icd/icd-10-cm/files.html
- CDC NCHS, icd10cm-Code Descriptions-2026.zip (icd10cm-codes-addenda-2026.txt) https://ftp.cdc.gov/pub/Health_Statistics/NCHS/Publications/ICD10CM/2026/
- HL7 FHIR R4 (4.0.1), Using ICD-[X] Codes with FHIR https://hl7.org/fhir/R4/icd.html; Code systems https://hl7.org/fhir/R4/terminologies-systems.html
- HL7 FHIR R4, Data type definitions https://hl7.org/fhir/R4/datatypes-definitions.html
- HL7 FHIR R4, Observation https://hl7.org/fhir/R4/observation.html
- HL7 FHIR publication directory https://hl7.org/fhir/directory.html
- HL7 Version 2.5.1, Chapter 7 Observation Reporting (as published by HL7 Europe) https://www.hl7.eu/HL7v2x/v251/std251/ch07.html
- Regenstrief Institute, LOINC 20563-3 https://loinc.org/20563-3/
- Regenstrief Institute, Download LOINC https://loinc.org/downloads/
- US National Library of Medicine, SNOMED CT International Edition https://www.nlm.nih.gov/healthit/snomedct/international.html
- The Unified Code for Units of Measure https://ucum.org/ucum
- ISO, ISO 3166 Country codes https://www.iso.org/iso-3166-country-codes.html; Glossary for ISO 3166 https://www.iso.org/glossary-for-iso-3166.html; ISO 3166-1:2020 https://www.iso.org/standard/72482.html
- IANA, Language subtag registration for TR https://www.iana.org/assignments/lang-subtags-templates/TR.txt; IANA Language Subtag Registry https://www.iana.org/assignments/language-subtag-registry/language-subtag-registry
- PostgreSQL 19 documentation, 8.6 Boolean Type https://www.postgresql.org/docs/current/datatype-boolean.html
- Board of Inquiry final report (1985), hosted by the FAA https://www.faa.gov/sites/faa.gov/files/2024-12/AirCanada143_C-GAUN.pdf

Chapter 9. Place

- National Geospatial-Intelligence Agency, "World Geodetic System 1984" (frame description submitted to the UN International Committee on GNSS, 2012) https://www.unoosa.org/pdf/icg/2012/template/WGS_84.pdf
- Same NGA document, References
- IOGP EPSG Geodetic Parameter Registry, CRS 4326 (API record and WKT export) https://apps.epsg.org/api/v1/CoordRefSystem/4326/; and https://apps.epsg.org/api/v1/CoordRefSystem/4326/export/?format=wkt&formatVersion=2
- OGC Definitions Server, CRS84 https://www.opengis.net/def/crs/OGC/1.3/CRS84
- RFC 7946, The GeoJSON Format (August 2016), section 3.1.1 https://www.rfc-editor.org/rfc/rfc7946.html
- RFC 7946, section 3.1.10
- RFC 7946, section 11.2

- Computed for this book from the WGS 84 defining parameters a = 6378137 m, 1/f = 298.257223563 (meridian and prime-vertical radii
- Computed for this book (haversine
- SCAR Composite Gazetteer of Antarctica (Antarctic Place Names), USA entry "Fuji Station, Dome" https://placenames.aq/api/place_names?place_name_gazetteer=ilike.*Fuji*
- SCAR Composite Gazetteer of Antarctica, USA entries "Queen Maud Land" and "Valkyrie Dome" https://placenames.aq/
- Apache Parquet format, LogicalTypes.md (GEOMETRY, GEOGRAPHY) and Geospatial.md https://github.com/apache/parquet-format/blob/master/LogicalTypes.md
- USPS, Publication 28, section 211 https://pe.usps.com/text/pub28/28c2_001.htm

Chapter 10. Absence

- Crossref record, DOI 10.1145/320107.320109 https://doi.org/10.1145/320107.320109
- Codd 1979, pp. 404 to 408 (scan of the published article), Computer History Museum archive https://archive.computerhistory.org/resources/access/text/2023/06/102659281-05-08-acc.pdf
- Codd 1979, p. 409, same scan
- SIGMOD Record archive PDF https://sigmodrecord.org/publications/sigmodRecord/8612/pdfs/16301.16303.pdf Crossref DOI 10.1145/16301.16303
- Codd 1986, section 1.1, pp. 55 to 58
- Codd 1986, section 2.7, pp. 75 to 76
- Codd 1986, p. 54
- Internet Archive catalogue record https://archive.org/details/relationalmodelf0000codd; publisher table of contents as listed by Schweitzer Fachinformationen https://www.schweitzer-online.de/buch/Codd/RELATIONAL-MODEL-DATABASE-MANAGEMENT-2/9780201141924/A27894773
- ISO catalogue https://www.iso.org/standard/76584.html
- PostgreSQL 19 Documentation, 9.1 Logical Operators https://www.postgresql.org/docs/current/functions-logical.html
- PostgreSQL 19 Documentation, 9.2 Comparison Functions and Operators https://www.postgresql.org/docs/current/functions-comparison.html
- PostgreSQL 19 Documentation, 9.21 Aggregate Functions https://www.postgresql.org/docs/current/functions-aggregate.html
- PostgreSQL 19 Documentation, 8.1 Numeric Types https://www.postgresql.org/docs/current/datatype-numeric.html
- Oracle Database SQL Language Reference (release 26), Nulls https://docs.oracle.com/en/database/oracle/oracle-database/26/sqlrf/Nulls.html the release 23 address now redirects there
- RFC 4180, section 2 https://www.rfc-editor.org/rfc/rfc4180.html
- RFC 8259 (December 2017), sections 3, 4 and 6 https://www.rfc-editor.org/rfc/rfc8259.html
- RFC 7396 (October 2014), section 1 https://www.rfc-editor.org/rfc/rfc7396.html
- Apache Parquet format, parquet.thrift, FieldRepetitionType https://github.com/apache/parquet-format/blob/master/src/main/thrift/parquet.thrift
- Apache Parquet format, README (Nested encoding; Nulls) https://github.com/apache/parquet-format/blob/master/README.md
- Apache Parquet format, parquet.thrift, Statistics, same URL
- ECMA-262, ECMAScript 2025 Language Specification, sections 3 and 6.1.6.1.13 https://tc39.es/ecma262/2025/
- pandas 3.1.1 documentation, pandas.read_csv https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html
- Computed for this book with pandas 2.2.3
- United Nations Statistics Division, Standard country or area codes for statistical use (M49) https://unstats.un.org/unsd/methodology/m49/overview/
- HL7 FHIR R4, CodeSystem data-absent-reason (http://terminology.hl7.org/CodeSystem/data-absent-reason;) https://hl7.org/fhir/R4/codesystem-data-absent-reason.html

Chapter 11. Records as Text: Fixed Width, CSV, XML and JSON

- RFC Editor, RFC 4180 https://www.rfc-editor.org/rfc/rfc4180.txt
- Python 3 documentation, csv: CSV File Reading and Writing https://docs.python.org/3/library/csv.html; Python changelog https://docs.python.org/3/whatsnew/changelog.html
- same page, footnote 1

- W3C, REC-xml-19980210 https://www.w3.org/TR/1998/REC-xml-19980210
- same document, sections 2.11 and 4.3.3
- W3C, REC-xmlschema-2-20010502 https://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
- W3C, XML Schema Part 1: Structures Second Edition (28 October 2004) https://www.w3.org/TR/xmlschema-1/
- W3C, XSD 1.1 Part 2: Datatypes https://www.w3.org/TR/xmlschema11-2/
- RFC Editor, RFC 8259 and info page https://www.rfc-editor.org/rfc/rfc8259.txt; https://www.rfc-editor.org/info/rfc8259
- Ecma International, ECMA-404 https://ecma-international.org/publications-and-standards/standards/ecma-404/
- RFC 8259, section 6 https://datatracker.ietf.org/doc/html/rfc8259#section-6
- RFC 8259, section 4 https://datatracker.ietf.org/doc/html/rfc8259#section-4
- RFC 8259, section 8.1
- RFC Editor, RFC 7493, sections 2.2, 2.3, 4.3 https://www.rfc-editor.org/rfc/rfc7493.txt
- CVE Program record CVE-2017-12635 (CNA: Apache) https://www.cve.org/CVERecord?id=CVE-2017-12635
- Author's computation, Node.js v22.20.0 and Python 3.13.3, 20 November 2026
- RFC Editor, RFC 3339 https://www.rfc-editor.org/rfc/rfc3339.txt
- JSON Schema Validation 2020-12 https://json-schema.org/draft/2020-12/json-schema-validation
- W3C, XML Schema Part 2: Datatypes Second Edition, 3.2.3 decimal and 4.3.12 fractionDigits https://www.w3.org/TR/xmlschema-2/
- Unicode Consortium, Microsoft vendor mapping CP1252.TXT https://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP1252.TXT

Chapter 12. Schemas That Travel

- ITU-T Recommendation X.208 https://www.itu.int/rec/T-REC-X.208
- ITU-T Recommendation X.209 https://www.itu.int/rec/T-REC-X.209
- ITU-T Recommendation X.680 https://www.itu.int/rec/T-REC-X.680
- ITU-T X.690 (02/2021), introduction and clause 7 https://www.itu.int/rec/T-REC-X.690
- ITU-T X.691 (02/2021), clause 7 https://www.itu.int/rec/T-REC-X.691
- RFC Editor, RFC 5280, sections 4.1 and 4.1.2.5 https://www.rfc-editor.org/rfc/rfc5280.txt
- Protocol Buffers documentation, Overview https://protobuf.dev/overview/
- Protocol Buffers documentation, Encoding https://protobuf.dev/programming-guides/encoding/
- Protocol Buffers documentation, Language Guide (proto 3) https://protobuf.dev/programming-guides/proto3/
- same guide, Updating A Message Type and Reserved Field Names
- Protocol Buffers documentation, Application Note: Field Presence https://protobuf.dev/programming-guides/field_presence/
- Protocol Buffers documentation, ProtoJSON Format https://protobuf.dev/programming-guides/json/
- Apache Avro Specification 1.12.0, Schema Resolution https://avro.apache.org/docs/1.12.0/specification/
- same specification, Logical Types
- same specification, Object Container Files, Single-object encoding, Binary Encoding
- JSON Schema, Specification https://json-schema.org/specification; Validation 2020-12 https://json-schema.org/draft/2020-12/json-schema-validation
- Confluent documentation, Schema Evolution and Compatibility https://docs.confluent.io/platform/current/schema-registry/fundamentals/schema-evolution.html
- Confluent documentation, Formats, Serializers and Deserializers https://docs.confluent.io/platform/current/schema-registry/fundamentals/serdes-develop/index.html
- AWS Glue Developer Guide, AWS Glue Schema registry https://docs.aws.amazon.com/glue/latest/dg/schema-registry.html

Chapter 13. Industry Messages

- HL7 Version 2+ specification (HL7 Europe publication), data type DTM https://hl7.eu/refactored/dtDTM.html
- HL7 FHIR R4, Version History https://hl7.org/fhir/R4/history.html
- HL7 FHIR, Publication (Version) History https://hl7.org/fhir/history.html
- HL7 FHIR v6.0.0-ballot5 index https://hl7.org/fhir/6.0.0-ballot5/; HL7 announcement of FHIR R6 early ballot opening, September 2026 cycle https://www.hl7.org/documentcenter/public/ballots/2026SEP/Announcements/2026Sep_HL7_Announcement_of_FHIR_R6_Early_Ballot_Opening_for_September_Ballot_Cycle.pdf

- HL7 FHIR R4 (4.0.1), Data Types https://hl7.org/fhir/R4/datatypes.html
- HL7 FHIR R5, Data Types https://hl7.org/fhir/R5/datatypes.html
- Swift, "Swift accepts community request to start ISO 20022 migration in March 2023" https://www.swift.com/news-events/news/swift-accepts-community-request-start-iso-20022-migration-march-2023
- Swift, ISO 20022 FAQs, "MT to ISO 20022 Conversion" https://www.swift.com/standards/iso-20022/iso-20022-faqs/mt-iso-20022-conversion
- Payments Canada, Lynx Corporate to Bank usage guideline, CustomerCreditTransferInitiationV09 (pain.001.001.09) https://www.payments.ca/sites/default/files/CustomerCreditTransferInitiationV09%28pain.001.001.09%29.pdf
- CBPRPlus-pacs.008.001.08 usage guideline, OneClearstream CBPR+ Customers (SR2025), generated by Swift MyStandards, 26 January 2026, pp. 3, 30, 64 to 65 https://www.clearstream.com/resource/blob/4151636/4ac4198a086ff16b60e07bfd462f25f0/pacs-008-2-data.pdf
- Payments Market Practice Group, "Hybrid Postal Address", version 1.10, 20 October 2025 (the version checked; version 1.12 of 5 March 2026 is also published on swift.com), pp. 5 and 7 https://www.ecb.europa.eu/paym/groups/shared/docs/f2b11-pmpg-hybrid-postal-address-v1.10-20oct2025-.pdf also on swift.com
- European Central Bank, "Reassessment of timeline for November 2026 TARGET Services releases", 28 August 2026 https://www.ecb.europa.eu/press/intro/news/html/ecb.mipnews260828.en.html; Swift, "Swift accepts community request to extend structured address migration for ISO 20022 payment messages" https://www.swift.com/news-events/news/swift-accepts-community-request-extend-structured-address-migration-iso-20022-payment-messages
- European Central Bank, "November 2026 TARGET Services releases confirmed with deployment shifted to 28 November 2026", 4 September 2026 https://www.ecb.europa.eu/press/intro/news/html/ecb.mipnews260904.en.html
- ACORD, Property and Casualty Data Standards https://www.acord.org/standards-architecture/acord-data-standards/Property_Casualty_Data_Standards; ACORD, P&C Carrier Standards Membership https://www.acord.org/membership-participation/programs-offerings/p-c-carrier-standards-membership
- 45 CFR 162.1102 (LII) https://www.law.cornell.edu/cfr/text/45/162.1102
- CMS, "Qualifiers for ICD-10 Diagnosis Codes on Electronic Claims" https://www.cms.gov/medicare/coding/icd10/downloads/icd10listservqualifiersfordiagnosiscodes.pdf; CMS 837P edit reference https://www.cms.gov/regulations-and-guidance/guidance/transmittals/downloads/r11120tn2.pdf
- XBRL International, XBRL 2.1 Recommendation, sections 4.6.2, 4.6.5, 4.7.2 https://www.xbrl.org/Specification/XBRL-2.1/REC-2003-12-31/XBRL-2.1-REC-2003-12-31+corrected-errata-2013-02-20.html
- SEC press release 2018-117 https://www.sec.gov/newsroom/press-releases/2018-117; SEC, Inline XBRL Filing of Tagged Data, Release 33-10514 https://www.sec.gov/files/rules/final/2018/33-10514.pdf
- ISO 9735-1:2002 https://www.iso.org/obp/ui#iso:std:iso:9735:-1:ed-2:v1:en; UNECE, UN/EDIFACT Message Design Guidelines https://unece.org/fileadmin/DAM/trade/untdid/texts/old/d424_d.htm; UNECE UNTDID D.24A https://service.unece.org/trade/untdid/d24a/trmd/impdef_c.htm
- ISO, ISO 20022-1:2026 https://www.iso.org/standard/20022-1; ISO 20022-1:2013 https://www.iso.org/standard/55005.html
- CBPRPlus-pacs.008.001.08 usage guideline (SR2025), Clearstream/MyStandards, 26 January 2026, URL above
- Clearstream, notice on the Swift Standards Release 2026 timeline (last updated 22 September 2026) https://www.clearstream.com/clearstream-en/res-library/connectivity/c26050-5582472 Swift's own notice (swift.com, "Swift accepts community request to extend structured address migration for ISO 20022 payment messages") is the announcement it reports
- ICAO Doc 9303 Part 3, 8th edition 2021, s.4.6 and s.6 table A https://www.icao.int/sites/default/files/publications/DocSeries/9303_p3_cons_en.pdf

Chapter 14. Values at Rest: Columnar Files and Archives

- Apache Parquet format, LogicalTypes.md, DECIMAL https://github.com/apache/parquet-format/blob/master/LogicalTypes.md
- Apache Parquet format, LogicalTypes.md, DATE, same URL
- Apache Parquet format, LogicalTypes.md, TIMESTAMP compatibility, same URL
- Apache Parquet format, parquet.thrift, enum Type https://raw.githubusercontent.com/apache/parquet-format/master/src/main/thrift/parquet.thrift
- parquet.thrift (LogicalType); LogicalTypes.md (VARIANT, GEOMETRY, GEOGRAPHY), URLs above
- Apache Arrow, Columnar Format, Data Types https://arrow.apache.org/docs/format/Columnar.html
- Apache Arrow, format/Schema.fbs https://raw.githubusercontent.com/apache/arrow/main/format/Schema.fbs
- Apache ORC, Types https://orc.apache.org/docs/types.html
- Author's computation, pyarrow 23.0.0, Python 3.13, 20 November 2026

- Author's computation, Python 3.13 zoneinfo (IANA tz database), 20 November 2026
- Library of Congress, Sustainability of Digital Formats, PDF/A-1 https://www.loc.gov/preservation/digital/formats/fdd/fdd000125.shtml
- Apache Parquet format, README https://github.com/apache/parquet-format/blob/master/README.md
- Apache Parquet format, parquet.thrift (FileMetaData field 5; ColumnMetaData field 8), URL above
- Apache Arrow, format/Schema.fbs, Timestamp comment, URL above
- Author's computation, pyarrow 23.0.0 column statistics, 20 November 2026

Chapter 15. The Model Reads and Writes Values

- OpenAI, tiktoken repository README https://github.com/openai/tiktoken
- OpenAI, tiktoken 0.9.0, tiktoken/model.py (MODEL_TO_ENCODING, MODEL_PREFIX_TO_ENCODING) https://github.com/openai/tiktoken/blob/0.9.0/tiktoken/model.py
- OpenAI, tiktoken_ext/openai_public.py https://github.com/openai/tiktoken/blob/0.9.0/tiktoken_ext/openai_public.py
- Computed for this book with tiktoken 0.9.0, Python 3.13.3
- Anthropic, Tool use with Claude, section When required parameters are missing https://platform.claude.com/docs/en/agents-and-tools/tool-use/overview
- OpenAI, Function calling guide https://developers.openai.com/api/docs/guides/function-calling
- OpenAI, Structured Outputs guide https://developers.openai.com/api/docs/guides/structured-outputs
- Model Context Protocol specification 2025-06-18, Tools https://modelcontextprotocol.io/specification/2025-06-18/server/tools
- Model Context Protocol specification 2026-07-28, Tools https://modelcontextprotocol.io/specification/2026-07-28/server/tools
- JSON Schema Validation, draft 2020-12 https://json-schema.org/draft/2020-12/json-schema-validation
- IETF, RFC 3339 https://www.rfc-editor.org/rfc/rfc3339
- OpenAI, Vector embeddings guide https://developers.openai.com/api/docs/guides/embeddings
- pgvector README https://github.com/pgvector/pgvector
- Cohere, Embed API reference https://docs.cohere.com/reference/embed
- Computed for this book in Python 3.13.3
- Computed for this book (identity, and a random test in Python 3.13.3 with NumPy
- IBM Support, ASRA 0C7 on a COMP-3 add https://www.ibm.com/support/pages/node/6145983; IBM Support, APAR PH16562 https://www.ibm.com/support/pages/apar/PH16562
- Illustrative thread, One Night, Every Value (hop 13, hop 14

Chapter 16. Type Contracts in 90 Days

- dbt Labs, Add data tests to your DAG https://docs.getdbt.com/docs/build/data-tests
- Bitol, Open Data Contract Standard, schema https://bitol-io.github.io/open-data-contract-standard/latest/schema/
- PostgreSQL documentation, COMMENT https://www.postgresql.org/docs/current/sql-comment.html
- PostgreSQL documentation, Appendix B.2, Handling of Invalid or Ambiguous Timestamps https://www.postgresql.org/docs/current/datetime-invalid-input.html
- Python, PEP 495, Local Time Disambiguation https://peps.python.org/pep-0495/
- Computed for this book with Python 3.13.3 zoneinfo
- 15 U.S.C. 260a https://www.law.cornell.edu/uscode/text/15/260a date computed with Python 3.13.3
- Apache Parquet, parquet.thrift https://github.com/apache/parquet-format/blob/master/src/main/thrift/parquet.thrift
- Apache Avro 1.12.0 Specification https://avro.apache.org/docs/1.12.0/specification/
- JSON Schema Validation, draft 2020-12, s.7.2.1 https://json-schema.org/draft/2020-12/json-schema-validation
- HL7 FHIR R4 (v4.0.1), Profiling https://hl7.org/fhir/R4/profiling.html
- HL7 FHIR R5 (5.0.0), Data Types https://hl7.org/fhir/R5/datatypes.html
- NAIC, Model Regulation 672 https://content.naic.org/sites/default/files/call_materials/MO672.pdf
- 45 CFR 164.508(c)(1) and (c)(2)(iii) https://www.law.cornell.edu/cfr/text/45/164.508
- Computed for this book with Python 3.13.3
- Illustrative thread, One Night, Every Value (hop 14, Ledger

Epilogue: The Contract Travels or the Meaning Does Not

- Illustrative thread, One Night, Every Value
- Author's opinion

Appendix A. The Type Contract: Template, Tests and Mappings

- Apache Parquet, Logical Types https://parquet.apache.org/docs/file-format/types/logicaltypes/
- Apache Parquet, LogicalTypes.md https://github.com/apache/parquet-format/blob/master/LogicalTypes.md
- Apache Parquet, parquet.thrift https://github.com/apache/parquet-format/blob/master/src/main/thrift/parquet.thrift
- Apache Avro 1.12.0 Specification https://avro.apache.org/docs/1.12.0/specification/
- JSON Schema Validation (2020-12) https://json-schema.org/draft/2020-12/json-schema-validation
- RFC 3339, Date and Time on the Internet: Timestamps https://www.rfc-editor.org/rfc/rfc3339
- RFC 9557, Date and Time on the Internet: Timestamps with Additional Information https://www.rfc-editor.org/rfc/rfc9557
- RFC 8259, The JavaScript Object Notation (JSON) Data Interchange Format https://www.rfc-editor.org/rfc/rfc8259
- PostgreSQL 19 Documentation, COMMENT https://www.postgresql.org/docs/current/sql-comment.html
- PostgreSQL 19 Documentation, Numeric Types https://www.postgresql.org/docs/current/datatype-numeric.html
- PostgreSQL 19 Documentation, Date/Time Types https://www.postgresql.org/docs/current/datatype-datetime.html
- HL7 FHIR, Publication (Version) History https://hl7.org/fhir/history.html
- HL7 FHIR R5, Data Types https://hl7.org/fhir/R5/datatypes.html
- HL7 FHIR R5, Using Codes in Resources https://hl7.org/fhir/R5/terminologies.html
- HL7 FHIR R5, Extensibility https://hl7.org/fhir/R5/extensibility.html
- HL7 FHIR R5, Resource https://hl7.org/fhir/R5/resource.html
- HL7 FHIR Extensions Pack 5.4.0, tz-code https://hl7.org/fhir/extensions/StructureDefinition-tz-code.html
- HL7 FHIR Extensions Pack 5.4.0, data-absent-reason https://hl7.org/fhir/extensions/StructureDefinition-data-absent-reason.html
- HL7 Terminology, CodeSystem data-absent-reason https://terminology.hl7.org/CodeSystem-data-absent-reason.html

Appendix B. Format Decision Table

- RFC 4180, Common Format and MIME Type for CSV Files https://www.rfc-editor.org/rfc/rfc4180
- RFC 8259, The JavaScript Object Notation (JSON) Data Interchange Format https://www.rfc-editor.org/rfc/rfc8259
- JSON Schema Validation (2020-12) https://json-schema.org/draft/2020-12/json-schema-validation
- W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, Recommendation 5 April 2012 https://www.w3.org/TR/xmlschema11-2/
- HL7 FHIR R5, Data Types https://hl7.org/fhir/R5/datatypes.html
- Apache Avro 1.12.0 Specification https://avro.apache.org/docs/1.12.0/specification/
- Protocol Buffers, Language Guide (proto3) https://protobuf.dev/programming-guides/proto3/
- Apache Parquet, parquet.thrift https://github.com/apache/parquet-format/blob/master/src/main/thrift/parquet.thrift
- Apache Arrow, Schema.fbs https://github.com/apache/arrow/blob/main/format/Schema.fbs
- Library of Congress, Sustainability of Digital Formats, PDF/A-3 https://www.loc.gov/preservation/digital/formats/fdd/fdd000360.shtml
- Library of Congress, Sustainability of Digital Formats, PDF/A-1b https://www.loc.gov/preservation/digital/formats/fdd/fdd000252.shtml
- Anthropic, Claude Platform documentation, Define tools https://platform.claude.com/docs/en/agents-and-tools/tool-use/implement-tool-use

Appendix C. Failure Library

- Final Report of the Board of Inquiry, Air Canada Boeing 767 C-GAUN, Gimli, Manitoba (Lockwood, April 1985), copy hosted by the FAA https://www.faa.gov/sites/faa.gov/files/2024-12/AirCanada143_C-GAUN.pdf
- GAO, Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia, GAO/IMTEC-92-26 https://www.gao.gov/products/imtec-92-26

- Federal Register 80 FR 24789, FR Doc. 2015-10066 https://www.govinfo.gov/content/pkg/FR-2015-05-01/html/2015-10066.htm
- Federal Register 83 FR 52131 https://www.govinfo.gov/content/pkg/FR-2018-10-16/html/2018-22152.htm
- Ziemann, Eren and El-Osta, Gene name errors are widespread in the scientific literature https://pmc.ncbi.nlm.nih.gov/articles/PMC4994289/
- Bruford et al., Guidelines for human gene nomenclature, Nature Genetics 52, 754 to 758 (2020) https://doi.org/10.1038/s41588-020-0669-3; HGNC https://rest.genenames.org/fetch/symbol/SEPTIN1
- Ariane 5 Flight 501 Failure, Report by the Inquiry Board https://esamultimedia.esa.int/docs/esa-x-1819eng.pdf
- NASA, Mars Climate Orbiter Mishap Investigation Board Phase I Report https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf

Appendix D. Glossary

- Unicode 18.0.0 core specification, chapter 2 https://www.unicode.org/versions/Unicode18.0.0/core-spec/chapter-2/
- RFC 7946, The GeoJSON Format https://www.rfc-editor.org/rfc/rfc7946
- EPSG Geodetic Parameter Registry, CRS 4326 (WKT export) https://apps.epsg.org/api/v1/CoordRefSystem/4326/export/?format=wkt
- IANA, Time Zone Database https://www.iana.org/time-zones
- Apache Parquet, Logical Types https://parquet.apache.org/docs/file-format/types/logicaltypes/
- ITU-R Recommendation TF.460-6 (02/2002) https://www.itu.int/rec/R-REC-TF.460-6-200202-I/en
- BIPM, CGPM 28 (2026) Resolution 5 https://www.bipm.org/committees/cg/cgpm/28-2026/resolution-5
- Oracle, Java SE 21 API, RoundingMode https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/math/RoundingMode.html; PostgreSQL Documentation, Numeric Types https://www.postgresql.org/docs/current/datatype-numeric.html
- SIX, ISO 4217 Currency Code List One (published 2026-09-17) https://www.six-group.com/dam/download/financial-information/data-center/iso-currrency/lists/list-one.xml
- UCUM specification, version 2.2 https://ucum.org/ucum
- Unicode, UAX #15 https://www.unicode.org/reports/tr15/
- Apache Avro 1.12.0 Specification https://avro.apache.org/docs/1.12.0/specification/

THE SERIES

Every Record Counts

Every Record Counts is a series about data: where each record was born, what it means, who can change it, what it costs and who reads it now, from the tally stick to the training set. This is its fifth book. Each book stands on its own.

Also by the author:

Every Record Counts	**From Tally to Token**: the record, its readers, and what each era broke
Every Record Counts	**The Model as Reader**: data for training, retrieval and agents
Every Record Counts	**From Warehouse to Lakehouse**: data on every platform, and the price of exit
Every Record Counts	**Data on Every Desk**: files, spreadsheets and the PC era
Every Pixel Matters	**From Bit to Pixel**: the whole stack, from the bit to the pixel
Every Pixel Matters	**From Code to Glass**: the rendering layer
Every Pixel Matters	**From Silicon to Screen**: who controls each layer, and what trust costs
Medallion	**From Bronze to Gold**: why data layers decide what AI can do

Juan Cardena

Enterprise AI & Agentic Systems Architect · blog.jcardena.com

Download the latest edition and the blank Type Contract at blog.jcardena.com/en/ebooks/what-a-value-means.

Every Record Counts · Book 5 · What a Value Means.