

Why Two Identical Strings Differ

Inspect characters and repair only the distinctions you mean to change

01

02

03

How to use this guide

People cleaning copied labels, names, identifiers or structured text without losing intentional content.

An original twelve-record text-repair case with canonical variants, fullwidth identifiers, Turkish letters, nonbreaking spaces, a joiner sequence and two deliberately unresolved records.

- Distinguish appearance, code points, bytes and user-perceived text units.
- Choose field-specific normalization and preservation rules.
- Produce a reversible repair ledger and quarantine ambiguous corruption.

Scope and limitations

- The fixture uses fictional text records and names; no person's identity or language is inferred.
- Normalization does not translate text, correct spelling or recover unknown original characters.
- The supplied checker diagnoses these explicit cases; it is not a complete security, font or grapheme-segmentation audit.
- No production database, original customer file or website content is changed.

Published by KitForma. Edition 1.0. Sources checked 1 October 2026. People and businesses in worked examples are fictional. This is an independent publication, not endorsed by the platforms mentioned.

Copyright 2026 KitForma. Usage, support and refund terms are included in LICENSE.txt in the purchased package.

Contents

How to use this guide	2
Contents	3
Describe the failure before changing the text	4
Sources and tools	6

01

Describe the failure before changing the text

Two labels can look alike while failing an exact comparison, and two different labels can become equal after an aggressive cleanup. Before choosing a repair, describe the actual failure. Is a search missing an expected result, an identifier failing a match, a line breaking in the wrong place, or an export displaying replacement symbols? Those are different problems. A global command to remove special characters may make one symptom disappear while damaging names, notation or identifiers elsewhere in the same file.

The original exercise is a twelve-record intake for the fictional Harbor Labels project. It mixes display labels, preserved identifiers, names, a unit label, an emoji sequence, a code literal and two records requiring review. The mixture is intentional: it makes a single universal cleaning rule visibly inappropriate. No records belong to real people or systems. Our goal is a repair candidate with a ledger showing which values changed, which stayed exact and which could not be safely resolved from the available evidence.

Keep the raw strings and their record IDs before inspecting them. A screenshot is not an adequate original because it captures appearance rather than the underlying characters. Save the source text in a known encoding, retain its context and record where the problem was observed. If a transformation goes wrong, the raw value lets you compare and recover. Without that copy, a later reviewer may be unable to distinguish an original character from one introduced during an attempted repair.

Set a narrow outcome. In this case, two records receive approved representation changes, eight are preserved and two are quarantined for

source review. The result is not a declaration that every unusual character was eliminated. Some unusual-looking text is correct, and some ordinary-looking text is ambiguous. A professional repair process respects both facts. It improves the behavior that was actually agreed without rewriting unrelated meaning merely to make the dataset look more uniform.

Sources and tools

[S1] Unicode Annex 15: Normalization Forms

<https://www.unicode.org/reports/tr15/>

Checked: 2026-10-01

[S2] Unicode Annex 29: Text Segmentation

<https://www.unicode.org/reports/tr29/>

Checked: 2026-10-01

[S3] Unicode FAQ: Case mappings and character properties

https://www.unicode.org/faq/casemap_charprop.html

Checked: 2026-10-01

[S4] W3C internationalization quick tips

<https://www.w3.org/International/quicktips/>

Checked: 2026-10-01

Unicode character inspector

Report code points and UTF-8 bytes for up to 5,000 code points; not a font or grapheme inspector.

<https://kitforma.com/en/unicode-inspector/>

Unicode normalizer

Compare explicit normalization forms while retaining the source; compatibility forms can alter meaningful distinctions.

<https://kitforma.com/en/unicode-normalizer/>

Text difference

Review text changes together with a character report when differences are visually subtle.

<https://kitforma.com/en/text-diff/>

Inside the complete guide

Inspect appearance, code points and bytes separately

Use canonical normalization for an approved purpose

Recognize compatibility changes before applying them

Treat case conversion as a separate operation

Inspect invisible characters in their field context

Know when the original cannot be reconstructed

Build a reversible transformation ledger

Capstone: repair the twelve Harbor records

Maintain precision without becoming afraid of text