QuickTranslate User Guide

Version 3.7.0 | February 2026 | LocaNext Project

> "Lookup & Transfer - Two Tools in One"

Table of Contents

QuickTranslate User Guide

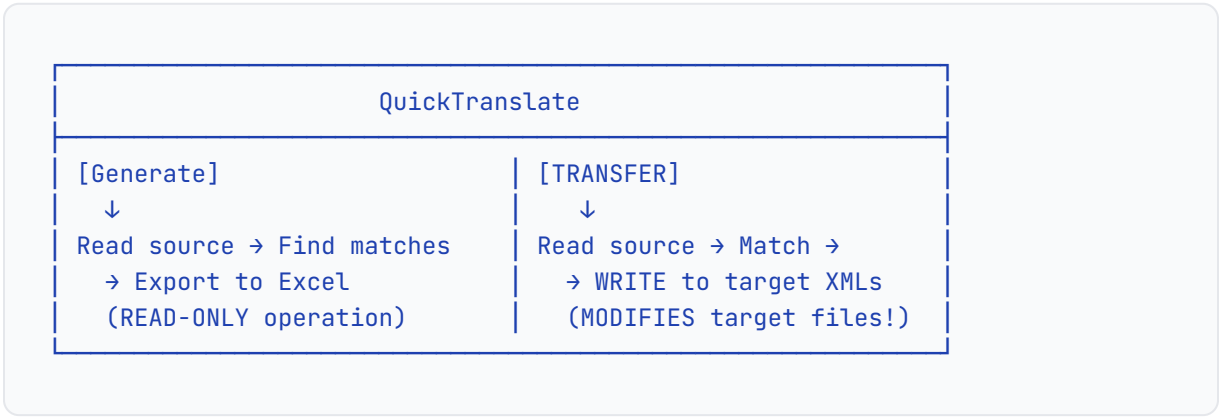
1. Introduction

1.1 What is QuickTranslate?

QuickTranslate is a dual-purpose desktop application for localization teams:

Function	Description
LOOKUP	Find translations of Korean text across 17 languages
TRANSFER	Write corrections from Excel/XML to target XML files

Two Buttons, Two Workflows



LOOKUP (Generate Button)

- Find translations for Korean text
- Look up any StringID to see all languages
- Reverse-lookup: find StringID from text in any language
- **Output:** Excel file with all translations

- **Safe:** Read-only, never modifies source files

TRANSFER (TRANSFER Button)

- Read corrections from Excel or XML
- Match corrections to target XML files
- **Write** corrections to target languagedata_*.xml files
- **Output:** Modified XML files in LOC folder
- **Careful:** Modifies target files!

1.2 Who is it for?

Role	LOOKUP Use Case	TRANSFER Use Case
Localization Coordinators	Find existing translations	Apply batch corrections
QA Testers	Verify translation consistency	Fix verified issues
Translators	Look up reference translations	Submit corrections
Developers	Find StringIDs from text	Update localization files

1.3 Key Benefits

Feature	LOOKUP	TRANSFER
Speed	Process hundreds of strings in seconds	Update multiple files at once
Accuracy	Multiple matching strategies	Strict and StringID-only modes
Completeness	Access all 17 languages	Target all languagedata files
Flexibility	Excel and XML input/output	Excel and XML corrections
Safety	Read-only operation	Confirmation before write

2. Installation

2.1 System Requirements

Requirement	Specification
Operating System	Windows 10 / Windows 11
Perforce Access	Sync access to stringtable folders
Drive	F: drive mapped (or custom path configured)
Python	3.11+ (portable version only)

2.2 Installation Methods

2.2.1 Setup Installer (Recommended)

1. Download `QuickTranslate_vX.X.X_Setup.exe` from releases
2. Run the installer
3. Select installation drive (C:, D:, F:, etc.)
4. Click **Install**
5. Application launches automatically

2.2.2 Portable Version

1. Download `QuickTranslate_vX.X.X_Portable.zip`
2. Extract to any folder

3. Run `QuickTranslate.exe`

2.3 First-Time Configuration

On first launch, QuickTranslate creates `settings.json`:

```
{
  "loc_folder": "F:\\perforce\\cd\\mainline\\resource\\GameData\\stringtable\\loc",
  "export_folder": "F:\\perforce\\cd\\mainline\\resource\\GameData\\stringtable\\export__"
}
```

To change paths: 1. Close QuickTranslate 2. Edit `settings.json` in the application folder 3. Update paths to match your Perforce workspace 4. Restart QuickTranslate

3. Quick Start

3.1 Your First LOOKUP (Translation Search)

Goal: Find translations for Korean strings

Step 1: Prepare Input Excel

Column A
안녕하세요
감사합니다
시작하기

Save as `input.xlsx`

Step 2: Configure

1. Launch QuickTranslate
2. Set **Format**: Excel
3. Set **Mode**: File
4. Set **Match Type**: Substring Match

Step 3: Select & Generate

1. Click **Browse** → select `input.xlsx`
2. Click **Generate**

Step 4: View Results

Output: `Output/QuickTranslate_YYYYMMDD_HHMMSS.xlsx`

KOR (Input)	ENG	FRE	GER	...
안녕하세요	Hello	Bonjour	Hallo	...

3.2 Your First TRANSFER (Apply Corrections)

Goal: Apply corrections from Excel to LOC XML files

Step 1: Prepare Corrections Excel

StringID	StrOrigin	Correction
UI_001	확인 버튼	OK Button (fixed)
UI_002	취소 버튼	Cancel Button (fixed)

Columns can be in any order - QuickTranslate auto-detects them.

Step 2: Configure

1. Set **Format**: Excel
2. Set **Mode**: File
3. Set **Match Type**: StringID + StrOrigin (STRICT)

Step 3: Select Files

1. **Source**: Browse → select your corrections Excel

2. **Target**: Browse → select LOC folder (or leave default)

Step 4: Transfer

1. Click **TRANSFER** (red button)
2. Confirm the operation in dialog
3. View results in log

Step 5: Verify

Check the modified `languagedata_*.xml` files in target folder.

3.3 Quick StringID Lookup

Goal: Find all translations for a specific StringID

1. Enter StringID in Quick Actions section (e.g., `UI_MainMenu_Title`)
2. Click **Lookup**
3. Output: Excel with all 17 language translations

3.4 Reverse Lookup

Goal: Find StringID from English (or any language) text

1. Create text file: `Start Game Options Exit`
2. In Quick Actions → Reverse, click **Browse**
3. Select your text file
4. Click **Find All**

5. Output: Excel with StringID and all translations

3.5 Find Missing Translations (Enhanced in v3.7.0)

Goal: Find Korean strings in TARGET that are MISSING from SOURCE — with 4 match modes, fuzzy matching, and category clustering.

Quick Start Flow

```
┌──────────────────────────────────────────────────────────┐
│  1. Set Source (corrections) and Target (LOC folder)     │
│  2. Click "Find Missing Translations" (purple button)    │
│  3. Popup appears → Choose match mode + threshold        │
│  4. Select output directory                              │
│  5. View results: Excel reports + Close folders per language  │
└──────────────────────────────────────────────────────────┘
```

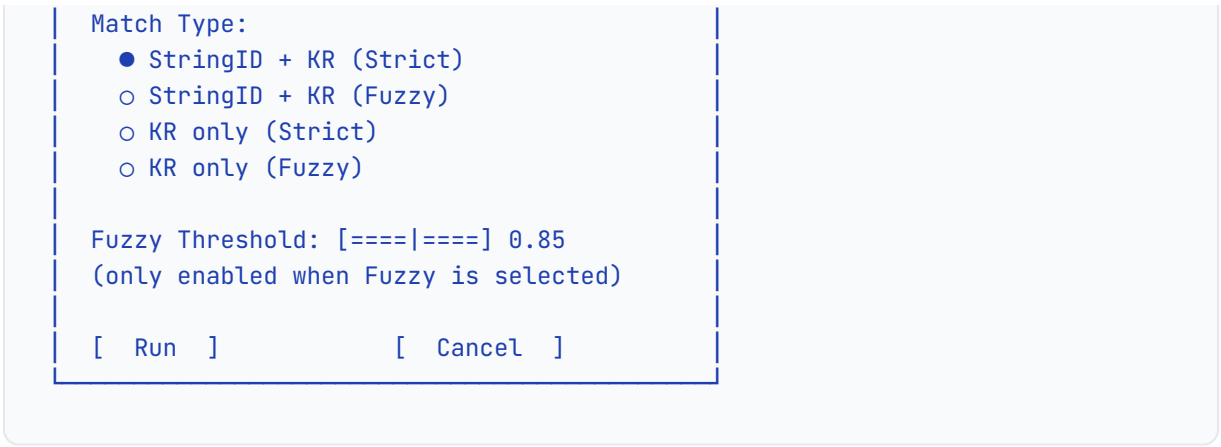
Step 1: Prepare Source & Target

- **Source:** Reference/corrections folder or file (contains keys you expect)
- **Target:** LOC folder with `languagedata_*.xml` files

Step 2: Click Find Missing

1. Click **Find Missing Translations** (purple button in Quick Actions)
2. A **parameter popup** appears:

```
┌──────────────────────────────────────┐
│  Find Missing Translations - Parameters  │
├──────────────────────────────────────┤
```



Step 3: Select Output & Run

1. Choose output directory when prompted
2. Watch detailed progress in terminal and GUI log area
3. Progress tracks: filtering, encoding, matching per language

Step 4: Review Output

Output	Description
Per-language Excel	`MISSING_{LANG}_{timestamp}.xlsx` — one per language with category clustering
Close folders	`Close_{LANG}/` — EXPORT-mirrored XML structure for direct re-import

> See Section 7: Find Missing Translations for full details on all 4 match modes, flowcharts, category clustering, and output format.

4. Core Concepts

4.1 StringID, StrOrigin, and Translations

StringID

Unique identifier for each localized string:

```
UI_MainMenu_Title_001
Quest_Chapter1_Dialog_042
Item_Weapon_Sword_Name
```

StrOrigin

Original Korean source text:

```
<LocStr StringId="UI_Button_OK" StrOrigin="확인" Str="OK" />
```

Translations

Stored in `languagedata_*.xml` files (17 languages).

4.2 SCRIPT Categories

SCRIPT categories have StrOrigin = raw Korean text:

Category	Content Type
Sequencer	Cutscene dialogue
AIDialog	NPC AI dialogue
QuestDialog	Quest conversations
NarrationDialog	Narrator/voiceover

Important: For SCRIPT strings, use **StringID-Only** match mode.

4.3 LOOKUP vs TRANSFER

Aspect	LOOKUP (Generate)	TRANSFER
Purpose	Find translations	Apply corrections
Output	Excel file	Modified XML files
Operation	Read-only	Writes to files
Confirmation	None needed	Required before write
Undo	N/A	Use Perforce revert

4.4 File Structure

LOC Folder (Target for TRANSFER)

```
loc/
├── languagedata_eng.xml
├── languagedata_fre.xml
├── languagedata_ger.xml
└── ... (17 files)
```

Export Folder (Source for LOOKUP)

```
export__/
├── Sequencer/
├── UI/
├── Items/
└── Quest/
```

5. LOOKUP Features

5.1 Generate Button

The **Generate** button performs read-only translation lookup:

```
Input (Korean text) → Match against stringtables → Output Excel
```

Input Modes

Mode	Description
File	Single Excel or XML file
Folder	All files in folder (recursive)

Format Modes

Format	Extensions	Use Case
Excel	.xlsx, .xls	Korean text in Column A
XML	.xml, .loc.xml	LocStr elements with StringId

Mixed File Support (Folder Mode)

When using Folder mode, QuickTranslate automatically detects and processes: - All `.xlsx` and `.xls` files - All `.xml` files

Files are combined into a single output.

5.2 StringID Lookup

Direct lookup of any StringID:

1. Enter StringID in the text field
2. Click **Lookup**
3. Get Excel with all 17 translations

Output columns: StringID | ENG | FRE | GER | SPA | ...

5.3 Reverse Lookup

Find StringID from text in ANY language:

1. Create text file with strings (one per line)
2. Click **Browse** → select file
3. Click **Find All**

Auto-detection: Identifies which language each input string is in.

Output columns: Input | KOR | ENG | FRE | GER | ...

5.4 ToSubmit Integration

Enable the checkbox to include files from `ToSubmit/` folder:

- Automatically loads correction files staged for submission
- Combines with selected source file/folder
- Useful for batch processing pending corrections

6. TRANSFER Features

6.1 TRANSFER Button

The **TRANSFER** button writes corrections to target XML files:

```
Corrections (Excel/XML) → Match in target → WRITE to languagedata_*.xml
```

Important Notes

1. **Confirmation Required:** Dialog asks for confirmation before writing
2. **Backup Recommended:** Use Perforce or manual backup before transfer
3. **Target Default:** LOC folder from settings.json

6.2 Source Formats

Excel Corrections

Required columns (auto-detected, case-insensitive): - **StringID** (or StringId, string_id) - **StrOrigin** (or Str_Origin, str_origin) - **Correction** (or correction)

Example: | StringID | StrOrigin | Correction | |----------|----------|----------| | UI_001 | 확인 버튼 | OK Button |

XML Corrections

Standard LocStr format:

```
<LocStr StringId="UI_001" StrOrigin="확인 버튼" Str="OK Button" />
```

Case-insensitive attributes: StringId, StringID, stringid, STRINGID all work.

6.3 Transfer Modes

File Mode

- Source: Single Excel or XML file
- Target: LOC folder or specific languagedata_*.xml

Language Detection: Extracts language code from filename: - `corrections_ENG.xlsx` → `languagedata_eng.xml` - `languagedata_FRE.xml` → `languagedata_fre.xml`

Folder Mode

- Source: Folder with multiple Excel/XML files organized by language
- Target: LOC folder (locdev__ or loc)

Smart Auto-Recursive Detection: QuickTranslate automatically detects language from folder structure:

```
TOSUBMIT/                          ← Source folder
├── 프랑스어_FRE/                  ← Language detected from suffix: FRE
│   ├── file1.loc.xml              ← All files → languagedata_FRE.xml
│   └── file2.loc.xml
├── 독일어_GER/                    ← Language detected from suffix: GER
│   └── correction.loc.xml         ← All files → languagedata_GER.xml
├── 포르투갈어_por-BR/             ← Language detected from suffix: POR-BR
│   └── update.loc.xml             ← All files → languagedata_POR-BR.xml
└── hotfix_SPA-ES.xml              ← Direct file with suffix → languagedata_SPA-ES.xml
```

Language Detection Rules: 1. **Folder suffix:** `FolderName_LANG/` → all files inside assigned to LANG 2. **File suffix:** `filename_LANG.xml` → single file assigned to LANG 3. **Hyphenated codes:** Supports `ZHO-CN`, `ZHO-TW`, `SPA-ES`, `SPA-MX`, `POR-BR` 4. **Case-insensitive:** `_fre`, `_FRE`, `_Fre` all work

Batch Processing: All corrections automatically routed to matching language files.

6.4 Match Modes for TRANSFER

STRICT Mode (Recommended)

Matches by **both** StringID AND StrOrigin: - Most precise - no false positives - Requires StrOrigin in corrections - Use for: General corrections

StringID-Only Mode

Matches by StringID only: - For SCRIPT categories (Sequencer, Dialog) - StrOrigin not required for matching - Use for: Dialogue corrections

6.5 Transfer Report

After transfer, the log shows:

```
═══════════════════════════════════════
     TRANSFER REPORT
═══════════════════════════════════════
● languagedata_eng.xml: 45 updated
● languagedata_fre.xml: 42 updated
○ languagedata_ger.xml: 0 updated (no matches)

Summary:
  Matched: 150
  Updated: 87
```

```
  Not Found: 12
═══════════════════════════════════════════
```

Symbols: - ● = Updates applied - ○ = No matches found - × = Error during processing

6.6 Folder Analysis

When you browse a **Source** or **Target** folder, QuickTranslate automatically analyzes the folder contents and prints a detailed summary to both the terminal and the GUI log.

What It Shows

Information	Description
File count	Total items, XML files, Excel files, other files, subdirectories
Languagedata index	Numbered table of all `languagedata_*.xml` files found
Language codes	Detected language code for each file (ENG, FRE, GER, etc.)
File sizes	Human-readable size for each file
Eligibility check	Whether the folder is eligible for TRANSFER operations

Terminal Output Example

```
============================================================
  SOURCE FOLDER ANALYSIS
============================================================
  Path: D:\locmerge\source
  Total items: 14 (12 XML, 0 Excel, 2 other, 0 subdirs)
------------------------------------------------------------

  LANGUAGEDATA FILES (12 found):
  #    Filename                          Lang     Size
```

```
  ---- ---------------------------------- -------- ----------
  1    languagedata_ENG.xml               ENG      4.2 MB
  2    languagedata_FRE.xml               FRE      3.8 MB
  3    languagedata_GER.xml               GER      3.9 MB
  ...

  VALIDATION:
  [OK] Eligible for TRANSFER (12 language files)
  [OK] Languages: ENG, FRE, GER, ...
============================================================
```

GUI Log Summary

The GUI log area shows a condensed version: - Folder path - Number of languagedata files found - List of detected languages - Eligibility status

Error Handling

If the folder cannot be fully analyzed (e.g., permission errors, unreadable files), the analysis gracefully reports the issue without blocking the operation.

6.7 Cross-Match Analysis

Before a **TRANSFER** executes in **Folder mode**, QuickTranslate performs a cross-match analysis. This prints a detailed pairing report to the terminal showing which source correction files will be applied to which target languagedata files.

What It Shows

Information	Description
Source count	Number of languagedata files in the source folder
Target count	Number of languagedata files in the target folder
Matched pairs	Source-to-target file pairings by language code
Unmatched files	Any files that could not be paired

Terminal Output Example

```
============================================================
  TRANSFER CROSS-MATCH ANALYSIS
============================================================
  Source: 12 languagedata files
  Target: 12 languagedata files
  Matched: 12 pairs
------------------------------------------------------------

  MATCHED PAIRS (12):
    languagedata_eng.xml                ⟶ languagedata_ENG.xml
    languagedata_fre.xml                ⟶ languagedata_FRE.xml
    languagedata_ger.xml                ⟶ languagedata_GER.xml
    languagedata_spa.xml                ⟶ languagedata_SPA.xml
    languagedata_por.xml                ⟶ languagedata_POR.xml
    languagedata_ita.xml                ⟶ languagedata_ITA.xml
    languagedata_rus.xml                ⟶ languagedata_RUS.xml
    languagedata_tur.xml                ⟶ languagedata_TUR.xml
    languagedata_pol.xml                ⟶ languagedata_POL.xml
    languagedata_jpn.xml                ⟶ languagedata_JPN.xml
    languagedata_zho-cn.xml             ⟶ languagedata_ZHO-CN.xml
    languagedata_zho-tw.xml             ⟶ languagedata_ZHO-TW.xml
============================================================
```

Why This Matters

The cross-match analysis helps verify that: 1. All expected language files are present in both source and target 2. File naming is consistent so pairings are correct 3. No corrections will be silently skipped due to missing target files

If any source files have no matching target, or vice versa, they are listed under an **UNMATCHED** section so you can investigate before the transfer proceeds.

6.8 Full Transfer Tree

When you click **TRANSFER** with folder mode, QuickTranslate displays a complete transfer tree showing every file that will be processed.

What It Shows

Information	Description
Languages detected	Number of unique languages found from folder/file suffixes
Languages ready	Languages with matching target files
Languages skipped	Languages without matching target files
Per-language breakdown	Every source file grouped by language
File sizes	Size of each source file
Target mapping	Which `languagedata_*.xml` each file will merge into

Terminal Output Example

```
╔══════════════════════════════════════════════════════════════╗
║  FULL TRANSFER TREE                                          ║
```

```
╔══════════════════════════════════════════════════════════════════════════════╗
║  Source: C:\Users\...\TOSUBMIT                                               ║
║  Target: F:\perforce\...\locdev__                                            ║
╠══════════════════════════════════════════════════════════════════════════════╣
║  Languages: 12 detected, 12 ready, 0 skipped                                 ║
║  Files: 224 total, 224 will transfer, 0 skipped                              ║
╠══════════════════════════════════════════════════════════════════════════════╣
║                                                                              ║
║  [OK] FRE: 22 files → languagedata_FRE.xml                                   ║
║      SOURCE FILE                                  SIZE       STATUS          ║
║      ──────────────────────────────────────────── ────────── ──────────      ║
║      프랑스어_FRE/ (22 files)                                                ║
║        ├─ itemequip_weapon.staticinfo.loc.xml   35 KB      → OK              ║
║        ├─ itemequip_armor.staticinfo.loc.xml    20 KB      → OK              ║
║        ├─ KnowledgeInfo_Skill.staticinfo.loc.xml 12 KB     → OK              ║
║                                                                              ║
║  [OK] GER: 16 files → languagedata_GER.xml                                   ║
║      SOURCE FILE                                  SIZE       STATUS          ║
║      ──────────────────────────────────────────── ────────── ──────────      ║
║      독일어_GER/ (16 files)                                                  ║
║        ├─ characterinfo_Animal.staticinfo.loc.xml  71 KB  → OK               ║
║        ...                                                                   ║
╚══════════════════════════════════════════════════════════════════════════════╝

Legend: [OK] = Ready to transfer  [!!] = No target (skipped)  [--] = Empty
```

Status Icons

Icon	Meaning
`[OK]`	Language has matching target file, ready to transfer
`[!!]`	No matching target file found, files will be SKIPPED
`[--]`	Language folder is empty
`→ OK`	Individual file will be transferred
`→ SKIP`	Individual file will be skipped (no target)

Why This Matters

The full transfer tree lets you verify: 1. **All languages detected:** Confirms folder naming is correct 2. **All files included:** No files silently missed 3. **Target mapping correct:** Each file goes to the right `languagedata_*.xml` 4. **Skip warnings visible:** Know before transferring if any files will be skipped

6.10 Transfer Scope Option (NEW in v3.4.0)

Choose which entries to transfer based on their current translation state.

Location in GUI

In the **Match Type** section, two radio buttons control transfer scope:

```
Transfer Scope:  ( ) ALL    ( ) Untranslated only
```

Options

Option	Description
ALL	Transfer all corrections regardless of existing translation
Untranslated only	Skip entries where target `Str` is not Korean (already translated)

How "Untranslated only" Works

1. QuickTranslate reads the target XML file
2. For each correction, checks if target `Str` contains Korean characters
3. If `Str` is Korean (untranslated): Apply the correction

4. If `Str` is NOT Korean (already translated): Skip and log as "Skipped: already translated"

When to Use Each Option

Scenario	Recommended Option
Fresh batch of corrections	ALL
Re-running corrections after partial success	Untranslated only
Overwriting existing translations intentionally	ALL
Filling in gaps only	Untranslated only

Example Log Output

```
Skipped: already translated (target has non-Korean text)
  StringID: UI_Button_001
  Target Str: "OK Button" (not Korean)
```

6.11 Language Auto-Discovery

QuickTranslate automatically discovers available languages from your LOC folder.

How It Works

1. Scans LOC folder for `languagedata_*.xml` files
2. Extracts language codes: `languagedata_SPA-ES.xml` → `SPA-ES`
3. Builds list of valid language codes for detection

4. Includes regional variants: `SPA-ES`, `SPA-MX`, `POR-BR`, etc.

Supported Languages (Auto-Discovered)

Code	Language
ENG	English
FRE	French
GER	German
ITA	Italian
JPN	Japanese
KOR	Korean
POL	Polish
POR-BR	Portuguese (Brazil)
RUS	Russian
SPA-ES	Spanish (Europe)
SPA-MX	Spanish (Latin America)
TUR	Turkish
ZHO-CN	Chinese Simplified
ZHO-TW	Chinese Traditional

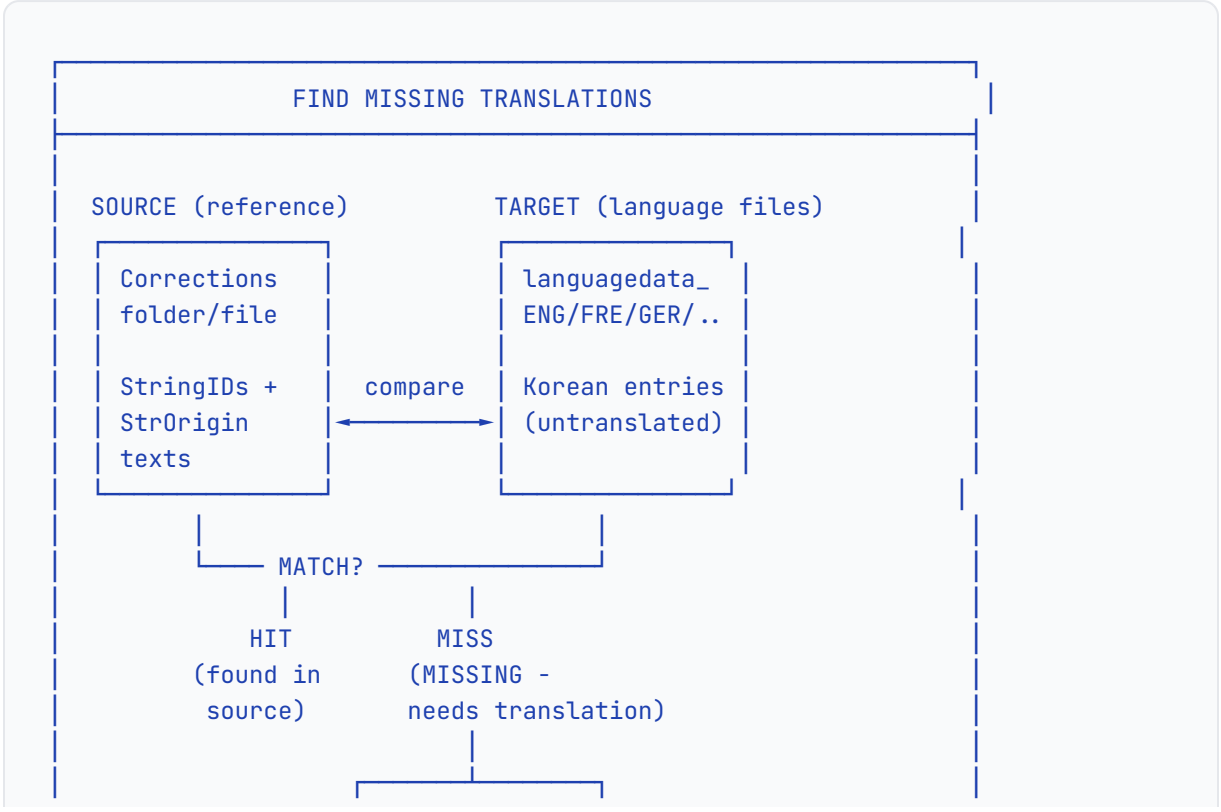
Note: Additional languages are automatically detected if present in your LOC folder.

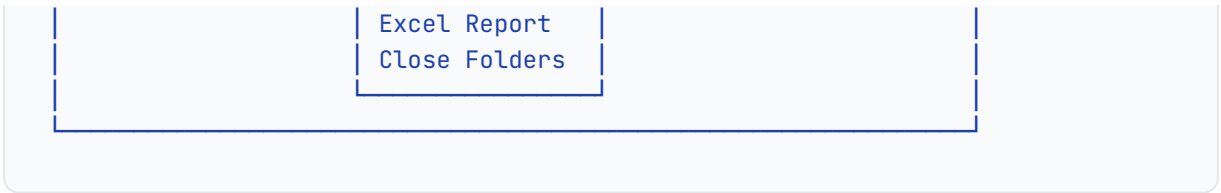
7. Find Missing Translations

> **NOTE**
>
> This is a major feature added in v3.5.0 and significantly enhanced in v3.7.0 with 4 match modes, KR-SBERT fuzzy matching, category clustering, and Close folder output.

7.1 Overview

Find Missing Translations identifies Korean (untranslated) strings in TARGET language files that are **not present** in your SOURCE reference. It answers the question: "Which strings still need translation?"





7.2 Match Mode Selection

When you click **Find Missing Translations**, a popup lets you choose from 4 match modes:

Mode	Key Used	Matching	Speed	Best For
StringID + KR (Strict)	(StrOrigin, StringID)	Exact	Instant	Default, most precise
StringID + KR (Fuzzy)	StringID groups	KR-SBERT cosine	Fast	Text rewording with same IDs
KR only (Strict)	StrOrigin text	Exact	Instant	Ignore StringID differences
KR only (Fuzzy)	All StrOrigin texts	KR-SBERT cosine	Slow	Most permissive, catches all

Mode Decision Guide

```
Do your corrections have the SAME StringIDs as the target?
  │
  ├─ YES → "StringID + KR" modes
  │    │
  │    ├─ Is the Korean text EXACTLY the same?
  │    │    ├─ YES → StringID + KR (Strict)    ← Recommended default
  │    │    └─ NO  → StringID + KR (Fuzzy)     ← Handles rewording
  │    │
  │
  └─ NO / UNSURE → "KR only" modes
```

```
        │
        ├─ Is the Korean text EXACTLY the same?
        │   ├─ YES → KR only (Strict)
        │   └─ NO  → KR only (Fuzzy)           ← Most permissive
```

7.3 How Each Mode Works

Mode 1: StringID + KR (Strict)

> **PRO TIP**
>
> This is the **RECOMMENDED DEFAULT**. Fastest and most precise.

Matching rule: Both StrOrigin AND StringID must match exactly.

```
SOURCE key: (StrOrigin="확인 버튼", StringID="UI_001")
TARGET key: (StrOrigin="확인 버튼", StringID="UI_001")
Result: HIT ✓  (both match exactly)

SOURCE key: (StrOrigin="확인 버튼", StringID="UI_001")
TARGET key: (StrOrigin="확인 버튼", StringID="UI_002")
Result: MISS ✗  (StringID differs)

SOURCE key: (StrOrigin="확인", StringID="UI_001")
TARGET key: (StrOrigin="확인 버튼", StringID="UI_001")
Result: MISS ✗  (StrOrigin text differs)
```

Data flow:

```
Step 1: Collect SOURCE composite keys → Set[(StrOrigin, StringID)]
Step 2: Filter TARGET to Korean-only entries (untranslated)
```

```
Step 3: For each Korean entry:
          if (entry.str_origin, entry.string_id) IN source_keys:
            → HIT (found, skip)
          else:
            → MISS (report as missing)
Step 4: Generate Excel + Close folders for MISS entries

ENCODING COST: Zero (pure set lookup, O(1) per entry)
```

Mode 2: StringID + KR (Fuzzy)

> **NOTE**
>
> Requires KR-SBERT model (`KRTransformer/` folder). Uses **pre-filtering by StringID** to minimize encoding cost.

Matching rule: StringID must exist in source. If yes, compare StrOrigin texts using KR-SBERT cosine similarity.

```
TARGET entry: StringID="UI_001", StrOrigin="확인 버튼을 누르세요"

Step 1: Is "UI_001" in SOURCE?
  NO  → INSTANT MISS (zero encoding!)
  YES → Continue to fuzzy comparison

Step 2: Encode TARGET text with KR-SBERT
Step 3: Compare against all SOURCE texts with StringID="UI_001":
  SOURCE group for UI_001:
    "확인 버튼"        → similarity 0.92  ← above threshold
    "버튼을 확인"      → similarity 0.88  ← above threshold

Step 4: max_similarity = 0.92 ≥ threshold (0.85) → HIT ✓
```

Pre-filtering flowchart:

```
┌──────────────────────────────────────────────────────────┐
│  STRINGID + KR (FUZZY) - Smart Filtering Pipeline        │
```

```
┌──────────────────────────────────────────────────────────┐
│                                                          │
│  SOURCE: 150,000 composite keys                          │
│    ↓                                                     │
│  Group by StringID:                                      │
│    "UI_001" → ["확인 버튼", "버튼을 확인"]                  │
│    "UI_002" → ["취소"]                                    │
│    ... (50K unique StringIDs, 1-5 texts each)            │
│    ↓                                                     │
│  Pre-encode ONLY grouped texts with KR-SBERT             │
│  Store: source_group_embeddings[StringID] = vectors      │
│                                                          │
│  TARGET: 23,000 Korean entries                           │
│    ↓                                                     │
│  For each entry:                                         │
│    StringID in source?                                   │
│      NO  → ★ INSTANT MISS (no encoding!) ★               │
│      YES → Encode 1 text, compare vs group               │
│            sim ≥ 0.85? → HIT                             │
│            sim < 0.85? → MISS (below threshold)          │
│                                                          │
│  Result: 2,456 INSTANT MISS + 1,500 BELOW THRESHOLD      │
│          = 3,956 total missing                           │
└──────────────────────────────────────────────────────────┘
```

Mode 3: KR only (Strict)

Matching rule: Only StrOrigin text must match. StringID is ignored entirely.

```
SOURCE texts: {"확인 버튼", "취소 버튼", "시작하기", ...}

TARGET entry: StrOrigin="확인 버튼", StringID="UI_999"
  "확인 버튼" IN source_texts? → YES → HIT ✓
  (StringID "UI_999" is completely ignored)

TARGET entry: StrOrigin="확인버튼", StringID="UI_001"
  "확인버튼" IN source_texts? → NO → MISS ✗
  (Space difference makes it a miss in strict mode)
```

> **NOTE**
>
> KR only (Strict) is **MORE PERMISSIVE** than StringID + KR (Strict) because it ignores StringID differences. Same text with different StringIDs will match.

Encoding cost: Zero (pure set lookup).

Mode 4: KR only (Fuzzy)

> **WARNING**
>
> This is the **SLOWEST MODE**. Encodes ALL source texts and ALL target texts with KR-SBERT. Use only when you need maximum permissiveness.

Matching rule: Find the most similar source text using cosine similarity. No StringID filtering.

```
┌──────────────────────────────────────────────────────────┐
│  KR ONLY (FUZZY) - Full Brute Force Pipeline             │
├──────────────────────────────────────────────────────────┤
│                                                          │
│  SOURCE: 120,000 unique StrOrigin texts                  │
│    ↓                                                     │
│  Encode ALL with KR-SBERT (batch 100)                    │
│  → source_embeddings: shape (120K, 768)                  │
│  → L2 normalize                                          │
│  TIME: ~5-10 minutes                                     │
│                                                          │
│  TARGET: 23,000 Korean entries                           │
│    ↓                                                     │
│  Process in batches of 100:                              │
```

```
│    Encode batch → (100, 768)                                  │
│    L2 normalize                                               │
│    Matrix multiply: (100, 768) × (768, 120K)                  │
│    = (100, 120K) similarity scores                            │
│    max_sim per row ≥ 0.85? → HIT / MISS                       │
│  TIME: ~2-5 minutes                                           │
│                                                               │
│  NO pre-filtering possible (no StringID constraint)           │
└───────────────────────────────────────────────────────────────┘
```

7.4 Performance Comparison

Mode	Source Encode	Target Encode	Shortcuts?	Time
StringID+KR Strict	None	None	Set lookup O(1)	**< 1 sec**
KR Strict	None	None	Set lookup O(1)	**< 1 sec**
StringID+KR Fuzzy	Per StringID group	Per entry (if SID found)	INSTANT MISS if SID not in source	**2-10 min**
KR Fuzzy	ALL source texts	ALL target texts (batched)	None possible	**10-20 min**

> **PRO TIP**
>
> Start with **STRINGID + KR (STRICT)** — it's instant and catches the most common cases. Only switch to fuzzy modes if you suspect text rewording.

7.5 The Filtering Pipeline

All modes share a common **pre-filtering** step that dramatically reduces the work:

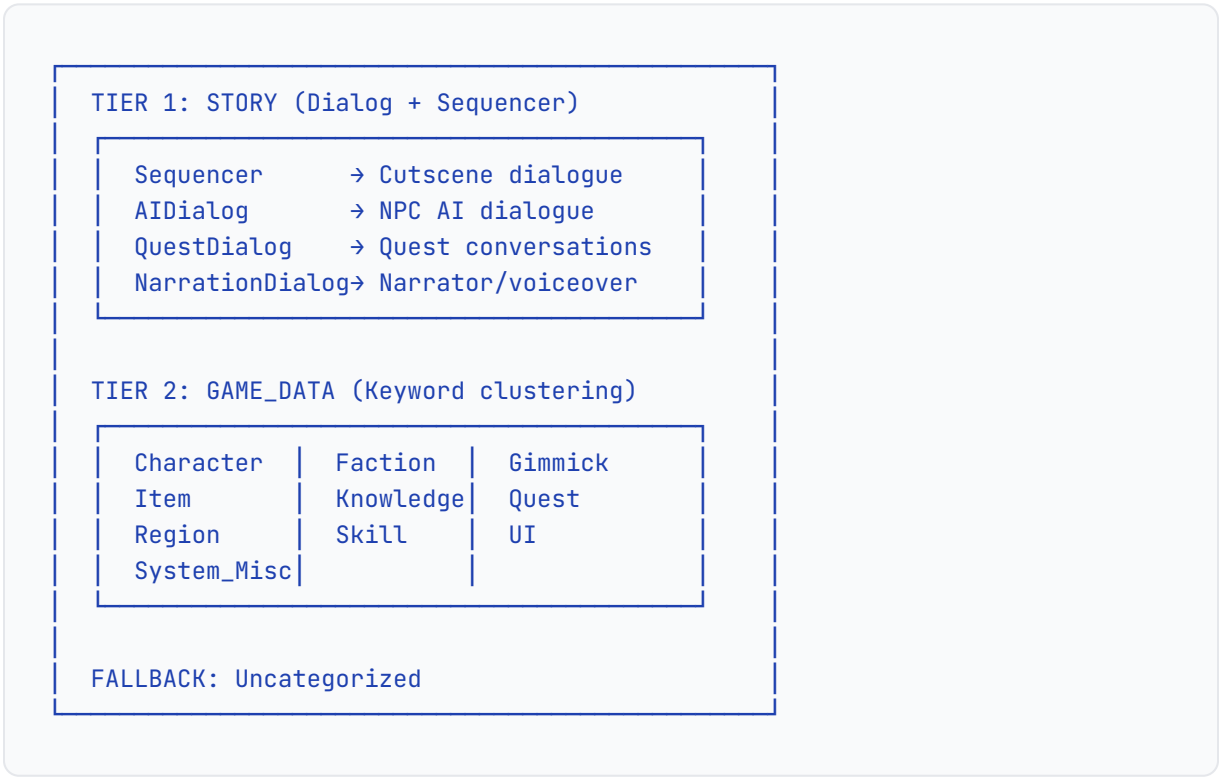
```
┌──────────────────────────────────────────────────────────┐
│  UNIVERSAL PRE-FILTER (all modes)                        │
├──────────────────────────────────────────────────────────┤
│                                                          │
│  Step 1: Scan TARGET languagedata_*.xml files            │
│                                                          │
│  Step 2: For each LocStr entry:                          │
│    Does Str contain Korean characters (Hangul)?          │
│      NO  → ★ SKIP (already translated) ★                 │
│      YES → Keep for matching                             │
│                                                          │
│  EXAMPLE:                                                │
│  <LocStr StringId="UI_001" StrOrigin="확인" Str="OK"/>   │
│    → Str="OK" has no Korean → SKIP (already translated)  │
│                                                          │
│  <LocStr StringId="UI_002" StrOrigin="취소" Str="취소"/>  │
│    → Str="취소" has Korean → KEEP (untranslated!)        │
│                                                          │
│  Result: Only UNTRANSLATED entries are processed         │
│          Typically 5K-50K per language (from 200K+)      │
└──────────────────────────────────────────────────────────┘
```

This means **no encoding is ever done on already-translated entries**. The universe is shrunk dramatically before any matching begins.

7.6 Category Clustering

Every missing entry is automatically categorized using the **EXPORT folder structure** — the same system used by LanguageDataExporter.

Category Hierarchy



How Categories Are Determined

The EXPORT folder is scanned at startup. Each `.loc.xml` file is categorized by:

1. **Top-level folder:** `Dialog/` → STORY, `Sequencer/` → Sequencer
2. **Priority keywords in filename:** `item`, `quest`, `skill`, `character`, etc.
3. **Standard folder patterns:** `UI/`, `Knowledge/`, `Faction/`, etc.
4. **Fallback:** `System_Misc`

Each StringID is mapped to exactly one category. This mapping is used in Excel reports.

Category Colors in Excel

Category	Color	Hex
Sequencer	Light Yellow	`#FFE599`
AIDialog	Light Green	`#C6EFCE`
QuestDialog	Light Green	`#C6EFCE`
NarrationDialog	Light Green	`#C6EFCE`
Item	Light Purple	`#D9D2E9`
Quest	Light Purple	`#D9D2E9`
Character	Light Orange	`#F8CBAD`
Gimmick	Light Purple	`#D9D2E9`
Skill	Light Purple	`#D9D2E9`
Knowledge	Light Purple	`#D9D2E9`
Faction	Light Purple	`#D9D2E9`
UI	Medium Green	`#A9D08E`
Region	Light Orange	`#F8CBAD`
System_Misc	Light Gray	`#D9D9D9`
Uncategorized	Beige	`#DDD9C4`

7.7 Output Files

Per-Language Excel Reports

Filename: `MISSING_{LANG}_{timestamp}.xlsx`

Each language gets its own Excel file with category-clustered missing entries:

Column	Content	Example
StrOrigin	Original Korean source text	확인 버튼을 누르세요
Translation	Current target text (Korean = untranslated)	확인 버튼을 누르세요
StringID	Unique string identifier	UI_Button_Confirm_001
Category	EXPORT-based category	UI

Sort order: STORY categories first (Sequencer, AIDialog, QuestDialog, NarrationDialog), then GAME_DATA alphabetically, then Uncategorized last.

Formatting: Auto-filter enabled, column widths auto-adjusted, category cells color-coded.

Close Folders (EXPORT-Mirrored Structure)

Folder: `Close_{LANG}/` (one per language with missing entries)

Close folders mirror the EXPORT folder structure, making it easy to re-import missing entries:

```
Output/
├── MISSING_FRE_20260206_120000.xlsx
├── MISSING_GER_20260206_120000.xlsx
├── Close_FRE/
│   ├── Dialog/
│   │   ├── AIDialog/
```

```
│   │   │   └── npc_greetings.loc.xml
│   │   └── QuestDialog/
│   │       └── quest_chapter1.loc.xml
│   ├── UI/
│   │   └── menu_strings.loc.xml
│   └── Item/
│       └── weapon_names.loc.xml
├── Close_GER/
│   ├── Dialog/
│   │   └── ...
│   └── ...
```

> **PRO TIP**
>
> Close folders can be
> **DIRECTLY USED AS CORRECTION INPUT**
>
> for the next TRANSFER operation. The EXPORT-mirrored structure ensures files go to the right categories.

7.8 Progress Tracking

The Find Missing Translations feature provides detailed progress tracking in both the **terminal** and **GUI log area**:

```
============================================================
FIND MISSING TRANSLATIONS - START
  Match mode: stringid_kr_fuzzy
  Threshold:  0.85
  Source:     D:\corrections\source
  Target:     F:\perforce\...\loc
  Output:     D:\output\missing_reports
  EXPORT:     F:\perforce\...\export__
============================================================
[Step 1] EXPORT indexes built: 45,293 categories, 45,293 paths
[Step 2] Collecting SOURCE keys (use_stringid=True, is_fuzzy=True)
[Step 2] SOURCE composite keys collected: 147,293
[Step 3] Scanning TARGET for Korean (untranslated) entries...
```

```
[Step 3] TARGET scan complete: 14 languages, 312,456 total Korean entries
  ENG: 23,456 Korean entries
  FRE: 22,890 Korean entries
  GER: 23,100 Korean entries
  ...
[Step 4] Preparing fuzzy embeddings with KR-SBERT...
[Step 4] stringid_kr_fuzzy: 48,293 unique StringID groups from 147,293 keys
  Encoding groups: 500/48,293 (1,234 texts)
  Encoding groups: 1,000/48,293 (2,567 texts)
  ...
[Step 4] Encoding complete: 48,293 groups, 152,000 total texts encoded
[Step 5] Finding MISSES per language...
  [1/14] Processing ENG: 23,456 Korean entries
    Fuzzy matching: 200/23,456 (HIT=180, SID_MISS=12, BELOW_THRESH=8)
    Fuzzy matching: 400/23,456 (HIT=365, SID_MISS=22, BELOW_THRESH=13)
    ...
    STRINGID_KR_FUZZY: 21,000 HIT, 1,456 StringID not in source, 1,000 below threshold
    Excel written: MISSING_ENG_20260206_120000.xlsx (2,456 rows)
  [2/14] Processing FRE: 22,890 Korean entries
    ...
[Step 6] Writing Close folders for 14 languages...
  Close_ENG/ written
  Close_FRE/ written
  ...
============================================================
FIND MISSING TRANSLATIONS - COMPLETE
  Total Korean entries: 312,456
  Total HITS (matched):  280,500
  Total MISSES:          31,956
  Languages processed:   14
  Excel files:           14
  Close folders:         14
============================================================
```

7.9 Use Cases

Scenario	Recommended Mode	Why
Pre-translation planning	StringID+KR Strict	Fast, precise count of untranslated strings
Progress tracking	StringID+KR Strict	Compare reports over time
Gap analysis	KR Strict	Find missing text regardless of StringID changes
After text rewording	StringID+KR Fuzzy	Catches similar but not identical Korean text
Maximum coverage	KR Fuzzy	Finds everything, even across different StringIDs
Re-import corrections	Any mode	Use Close folders as TRANSFER source

7.10 Fuzzy Threshold Guide

The threshold (0.00 - 1.00) controls how similar two Korean texts must be for a fuzzy match:

Threshold	Sensitivity	Example Match
1.00	Exact only	"확인 버튼" ↔ "확인 버튼" only
0.95	Very strict	Minor whitespace/punctuation differences
0.85	Recommended	"확인 버튼을 누르세요" ↔ "확인 버튼 누르기"
0.75	Permissive	"무기를 장착하세요" ↔ "장비를 착용하세요"
0.60	Very loose	May produce false positives

> **PRO TIP**
>
> Start with the default threshold of **0.85**. If too many false misses, lower to 0.80. If too many false hits, raise to 0.90.

8. Match Types

8.1 Substring Match (Original)

How it works:

```
Input: "시작"
Finds: Any string containing "시작"
  - "게임 시작하기" ✓
  - "시작 버튼" ✓
  - "새로 시작" ✓
```

Pros	Cons
Flexible	May return multiple matches
Finds partial matches	Less precise

Best for: Finding strings when you only have partial Korean text

Button: Generate (LOOKUP only)

8.2 StringID-Only (SCRIPT)

How it works: 1. Reads StringIDs from input 2. Filters to SCRIPT categories ONLY (Sequencer, AIDialog, QuestDialog, NarrationDialog) 3. Returns/applies translations for matching StringIDs

Status output: "SCRIPT filter: 150 kept, 23 skipped"

Best for: Processing Sequencer/Dialog corrections

Buttons: Generate (LOOKUP) and TRANSFER

8.3 StringID + StrOrigin (STRICT)

How it works:

```
Input: StringID="UI_001", StrOrigin="확인"
Matches: ONLY if both StringID AND StrOrigin match exactly
```

Pros	Cons
Most precise	Requires both fields
No false positives	More input data needed
Handles reused StringIDs	-

Best for: Verifying corrections with 100% certainty

Buttons: Generate (LOOKUP) and TRANSFER

8.4 Quadruple Fallback (Fuzzy KR Match)

How it works:

Uses KR-SBERT semantic similarity + FAISS to find matches when exact matching fails. Requires the `KRTransformer/` model folder alongside the app.

```
For each correction:
  L1 (HIGH):        StrOrigin + file_relpath + adjacency_hash  → exact context match
  L2A (MEDIUM-HIGH): StrOrigin + file_relpath                  → same file match
  L2B (MEDIUM):      StrOrigin + adjacency_hash                → same context match
```

```
L3 (LOW):          StrOrigin only                              → text-only match

If no exact match at any level → FAISS semantic similarity search
```

FAISS Technical Details: - **Model:** `snunlp/KR-SBERT-V40K-klueNLI-augSTS` (768-dim, local `KRTransformer/` folder) - **Index:** `faiss.IndexFlatIP` (inner product after L2 normalization = cosine similarity) - **Encoding:** Batch processing, 100 texts per batch - **Search:** One query at a time (same as TFM FULL, XLSTransfer, KR Similar monoliths) - **Threshold:** Configurable 0.80 - 1.00 (default 0.85)

Process flow: 1. Load KR-SBERT model (cached after first load) 2. Scan target folder for XML entries 3. Encode all StrOrigin texts in batches of 100 4. Build FAISS IndexFlatIP index (instant) 5. For each correction, search index for best match 6. Apply 4-level cascade with context disambiguation

Pros	Cons
Handles text variations	Requires KRTransformer model (~447MB)
Context-aware disambiguation	Slower than exact matching
Configurable threshold	First run builds index

Best for: Corrections where StrOrigin text may have minor differences from target

Buttons: TRANSFER (with Quadruple Fallback match type selected)

8.5 Special Key Match

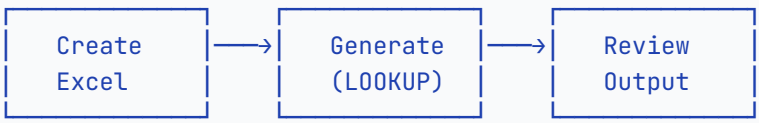
How it works: - Custom composite key from multiple fields - Configure key fields in the UI (comma-separated) - Default: `string_id,category`

Best for: Advanced matching scenarios

Buttons: Generate (LOOKUP) only

9. Workflows

9.1 LOOKUP: Find Translations



1. Create Excel with Korean text in Column A
2. Set Format: Excel, Mode: File, Match Type: Substring
3. Browse to file → Click **Generate**
4. Open output Excel

9.2 LOOKUP: Process SCRIPT Corrections

1. Set Format: XML
2. Set Match Type: StringID-Only (SCRIPT)
3. Browse to XML file
4. Click **Generate**

Output: Only SCRIPT categories included

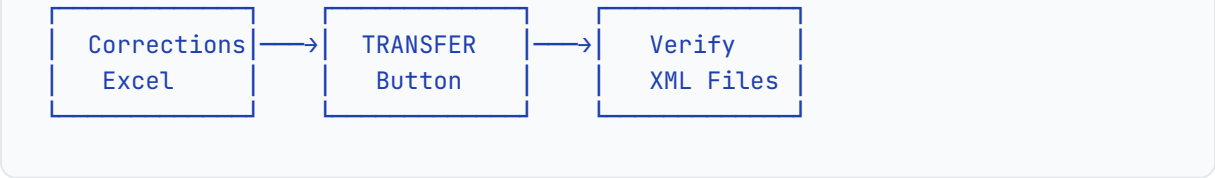
9.3 LOOKUP: Verify with Strict Matching

1. Set Format: XML

2. Set Match Type: StringID + StrOrigin (STRICT)
3. Browse to Source XML
4. Browse to Target folder
5. Click **Generate**

Output: Only verified matches

9.4 TRANSFER: Apply Excel Corrections



1. Prepare corrections Excel (StringID, StrOrigin, Correction)
2. Set Format: Excel, Mode: File
3. Set Match Type: STRICT (recommended)
4. Source: Browse to corrections file
5. Target: Browse to LOC folder
6. Click **TRANSFER** → Confirm
7. Check log for results

9.5 TRANSFER: Batch Apply from Folder

1. Set Mode: Folder
2. Set Match Type: STRICT or StringID-Only
3. Source: Browse to folder with corrections
4. Target: LOC folder

5. Click **TRANSFER** → Confirm
6. View transfer report

9.6 TRANSFER: SCRIPT Dialogue Corrections

For Sequencer/Dialog corrections:

1. Set Match Type: StringID-Only (SCRIPT)
2. Source: Corrections file
3. Target: LOC folder
4. Click **TRANSFER**

Note: Only SCRIPT categories are processed; others skipped.

10. Output Files

10.1 LOOKUP Outputs

All saved to: `<app_folder>/Output/`

Translation Output

Filename: `QuickTranslate_YYYYMMDD_HHMMSS.xlsx`

Column	Content
A	KOR (Input)
B	ENG
C-Q	Other languages...

Multiple matches: Formatted as numbered list

```
1. Translation option 1
2. Translation option 2
```

StringID Lookup Output

Filename: `StringID_<ID>_YYYYMMDD_HHMMSS.xlsx`

Single row with StringID and all translations.

Reverse Lookup Output

Filename: `ReverseLookup_YYYYMMDD_HHMMSS.xlsx`

Input	KOR	ENG	FRE	GER
Start Game	게임 시작	Start Game	Démarrer	Spiel starten

Special values: - `NOT FOUND` - No matching StringID - `NO TRANSLATION` - Translation empty

10.2 TRANSFER Outputs

Output: Modified `languagedata_*.xml` files in target folder

Log Report: Shown in application log area with: - Files processed - Matches found - Updates applied - Errors encountered

10.2.1 Failure Reports (Automatic)

When transfer has failures (not found, skipped), detailed reports are automatically generated:

Saved to: Source folder (same location as your corrections)

Per-Language Failure XML Files (NEW in v3.4.0)

Filename: `FAILED_<LANG>_YYYYMMDD_HHMMSS.xml` (e.g., `FAILED_FRE_20260205_120000.xml`)

Separate XML files are generated **per language** with clean LocStr format:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <LocStr StringId="UI_001" StrOrigin="확인" Str="OK" Category="UI"/>
```

```
  <LocStr StringId="UI_002" StrOrigin="취소" Str="Annuler" Category="UI"/>
</root>
```

Key features: - **Exact attribute preservation:** All original attributes (StringId, StrOrigin, Str, Category, etc.) are preserved exactly as-is - **No metadata:** Clean format without FailReason, SourceFile, or timestamp attributes on LocStr elements - **Per-language separation:** Each language gets its own file (FAILED_FRE, FAILED_GER, FAILED_SPA, etc.) - **Direct re-import:** Files can be used directly as correction input for the next transfer attempt

Use case: Re-import failed entries directly back into the correction workflow without manual cleanup

Combined Failure Report (Legacy)

Filename: `FAILED_TO_MERGE_YYYYMMDD_HHMMSS.xml`

Groups all failed LocStr entries by source file with metadata:

```
<?xml version="1.0" encoding="utf-8"?>
<FailedMerges timestamp="2026-02-05T14:30:00" total="45">
  <SourceFile name="corrections_fre.xml" language="FRE" failed="12">
    <LocStr StringId="UI_001" StrOrigin="확인" Str="OK"
            FailReason="StringID not found in target"/>
    ...
  </SourceFile>
</FailedMerges>
```

Use case: Detailed analysis of failures with reason tracking

Excel Failure Report

Filename: `FailureReport_YYYYMMDD_HHMMSS.xlsx`

4-sheet workbook with comprehensive analysis:

Sheet	Content
Summary	Total counts, success/failure rates, breakdown
Failure by Reason	Grouped by failure category with counts
Failure by File	Per-file statistics and success rates
Detailed Failures	Every failed entry with StringID, StrOrigin, reason

Failure reasons tracked: - `StringID not found in target` - ID doesn't exist in target XML - `Skipped: non-SCRIPT category` - StringID-Only mode filters non-Dialog/Sequencer - `Skipped: already translated` - "Untranslated only" mode skips non-Korean entries - `Skipped: excluded subfolder` - In exclusion list (narration etc.)

Use case: Analyze why corrections failed, prioritize fixes

11. Troubleshooting

11.1 LOOKUP Issues

"LOC folder not found"

Cause: Perforce not synced or path incorrect **Solution:** 1. Run `p4 sync` on stringtable folder 2. Or update `settings.json` with correct path

"Sequencer folder not found"

Cause: Export folder not synced **Solution:**

```
p4 sync //depot/cd/mainline/resource/GameData/stringtable/export__/...
```

"No input data found"

Cause: Empty file or wrong format **Solution:** - Excel: Ensure data is in Column A - XML: Ensure file has `<LocStr>` elements

"StringID not found"

Cause: StringID doesn't exist in current branch **Solution:** 1. Check spelling (case-sensitive) 2. Try different branch

11.2 TRANSFER Issues

"Source not found"

Cause: File path incorrect **Solution:** Use Browse button to select file

"Target folder not found"

Cause: LOC folder path incorrect **Solution:** Update `settings.json` or browse to correct folder

"0 matches found"

Causes: 1. StringID not in target files 2. StrOrigin doesn't match (STRICT mode) 3. Category not in SCRIPT set (StringID-Only mode)

Solutions: 1. Verify StringIDs exist in target 2. Check StrOrigin text matches exactly 3. Use STRICT mode for non-SCRIPT strings

"STRORIGIN_MISMATCH" (NEW in v3.4.0)

Cause: StringID exists in target XML but StrOrigin text doesn't match (STRICT mode only)

Message: `StrOrigin mismatch (StringID exists but source text differs)`

Why this happens: - The Korean source text (StrOrigin) was updated in the game data - Your correction file has an older version of the StrOrigin - StringID is correct, but the exact tuple match fails

Solutions: 1. Update your correction file with the current StrOrigin from target 2. Use **StringID-Only mode** if StrOrigin matching is not required 3. Check if the StrOrigin has special characters or whitespace differences

Example:

```
Correction: StringID="UI_001", StrOrigin="확인 버튼"
Target:     StringID="UI_001", StrOrigin="확인버튼"     <- space removed
Result:     STRORIGIN_MISMATCH (StringID found, StrOrigin differs)
```

"Transfer completed but file unchanged"

Cause: Corrections already applied or no differences **Solution:** This is normal if translations are identical

11.3 Fuzzy Matching Issues

"KRTransformer model not found"

Cause: The KR-SBERT model folder is missing **Solution:** Place the `KRTransformer/` folder alongside the app. Copy from `models/kr-sbert.deprecated/`

"sentence-transformers is not installed"

Cause: ML dependencies not installed **Solution:** `pip install -r requirements-ml.txt`

Fuzzy matching is slow

Expected performance: ~10K texts should encode in a few seconds. If it takes minutes: 1. Check you're using IndexFlatIP (not HNSW) - see `docs/FAISS_IMPLEMENTATION.md` 2. Ensure batch encoding is working (batch_size=100, not all-at-once) 3. First load of KR-SBERT model takes 5-10 seconds (cached after)

"Batches: 100% 1/1" spam in terminal

Cause: `show_progress_bar=False` missing on a `model.encode()` call **Solution:** Every `model.encode()` call must have `show_progress_bar=False`

11.4 Performance Tips

Scenario	Tip
First run slow	Building index + loading model. Subsequent runs faster (cached)
Large corrections file	Use Folder mode for batching
Memory usage	Close other apps for 1000+ corrections
Fuzzy matching slow	Ensure IndexFlatIP is used (see FAISS_IMPLEMENTATION.md)
Model loading	First load ~5-10s, then instant (cached in memory)

12. Reference

12.1 Supported Languages

Code	Display	Language
`kor`	KOR	Korean (Source)
`eng`	ENG	English
`fre`	FRE	French
`ger`	GER	German
`spa`	SPA	Spanish
`por`	POR	Portuguese
`ita`	ITA	Italian
`rus`	RUS	Russian
`tur`	TUR	Turkish
`pol`	POL	Polish
`zho-cn`	ZHO-CN	Chinese (Simplified)
`zho-tw`	ZHO-TW	Chinese (Traditional)
`jpn`	JPN	Japanese
`tha`	THA	Thai
`vie`	VIE	Vietnamese
`ind`	IND	Indonesian
`msa`	MSA	Malay

12.2 Supported File Formats

Format	Extensions	Library
Excel	`.xlsx`, `.xls`	openpyxl
XML	`.xml`, `.loc.xml`	lxml
Text	`.txt`	built-in

12.3 SCRIPT Categories

Category	Description
Sequencer	Cutscene/cinematic dialogue
AIDialog	NPC AI-triggered dialogue
QuestDialog	Quest conversation text
NarrationDialog	Narrator/voiceover text

12.4 Default Paths

Path	Default Value
LOC Folder	`F:\perforce\cd\mainline\resource\GameData\stringtable\loc`
Export Folder	`F:\perforce\cd\mainline\resource\GameData\stringtable\export__`
Output Folder	`<app_folder>\Output`
ToSubmit Folder	`<app_folder>\ToSubmit`
Settings File	`<app_folder>\settings.json`

12.5 Excel Column Detection

QuickTranslate auto-detects these column names (case-insensitive):

Column Purpose	Accepted Names
StringID	StringID, StringId, string_id, STRINGID
StrOrigin	StrOrigin, Str_Origin, str_origin, STRORIGIN
Correction	Correction, correction, Str, str

12.6 Command Line Options

```
python main.py              # Launch GUI
python main.py --verbose    # Launch with debug logging
```

```
python main.py --version    # Show version
python main.py --help       # Show help
```

12.7 Keyboard Shortcuts

Shortcut	Action
`Alt+G`	Generate
`Alt+T`	Transfer
`Alt+C`	Clear fields
`Alt+X`	Exit

13. Appendix

13.1 Glossary

Term	Definition
StringID	Unique identifier for a localized string
StrOrigin	Original Korean source text
Str	Translated text for a language
LocStr	XML element containing string data
SCRIPT	Categories with raw Korean StrOrigin
LOC folder	Contains `languagedata_*.xml` files
LOOKUP	Read-only translation search (Generate button)
TRANSFER	Write corrections to XML files (TRANSFER button)
FAISS	Facebook AI Similarity Search - vector index for fuzzy matching
IndexFlatIP	FAISS index type using inner product (cosine similarity)
KR-SBERT	Korean Sentence-BERT model for semantic text encoding
Quadruple Fallback	4-level cascade matching: context → file → adjacency → text
Find Missing	Feature to identify untranslated Korean strings across languages
Match Mode	Algorithm for comparing source vs target (Strict or Fuzzy, with/without StringID)
Category Clustering	Two-tier EXPORT-based classification (STORY + GAME_DATA)
Close Folder	Output directory mirroring EXPORT structure for easy re-import
Cosine Similarity	Measure of text similarity (0.0 = unrelated, 1.0 = identical)

Term	Definition
Fuzzy Threshold	Minimum cosine similarity for a fuzzy match (default 0.85)
Korean Filter	Pre-filter that only processes entries with Korean Str (untranslated)

13.2 XML Element Structure

```
<LocStr
    StringId="UI_Button_001"
    StrOrigin="확인 버튼"
    Str="OK Button"
    Category="UI"
/>
```

Attribute	Required	Description
StringId	Yes	Unique identifier
StrOrigin	No	Korean source text
Str	Yes	Translation text
Category	No	String category

13.3 Changelog

Version 3.7.0 (February 2026)

Major Feature: Find Missing Translations Enhancement

- **4 Match Modes:** StringID+KR Strict, StringID+KR Fuzzy, KR Strict, KR Fuzzy
- **Parameter Popup:** GUI dialog to select match mode and fuzzy threshold before running
- **Category Clustering:** EXPORT-based two-tier categorization (STORY + GAME_DATA) — same system as LanguageDataExporter
- **Per-Language Excel Reports:** `MISSING_{LANG}_{timestamp}.xlsx` with StrOrigin, Translation, StringID, Category columns, color-coded by category
- **Close Folders:** `Close_{LANG}/` output mirroring EXPORT directory structure for direct re-import as corrections
- **Fuzzy Matching:** KR-SBERT semantic similarity with configurable threshold (0.00-1.00)
- **Smart Pre-Filtering:** StringID+KR Fuzzy groups by StringID first, only encodes matching groups (INSTANT MISS for unknown StringIDs)
- **Detailed Progress Tracking:** 46 logger calls with [Step N] prefixes, batch progress (500/12000, 1000/12000...), per-language HIT/MISS/SID_MISS/BELOW_THRESH counters
- **GUI Log Bridge:** All progress messages appear in both terminal AND GUI log area

Technical: - New `core/category_mapper.py` — ported from LanguageDataExporter's TwoTierCategoryMapper - New `gui/missing_params_dialog.py` — Tkinter Toplevel popup for parameter selection - Enhanced `core/missing_translation_finder.py` — `find_missing_with_options()` function with 4 modes - Enhanced `gui/app.py` — progress_cb bridges to both `_update_status()` and `_log()` - Uses xlsxwriter for Excel output (not openpyxl) - Pure numpy for fuzzy matching (no FAISS dependency)

Version 3.6.0 (February 2026)

Critical Fix: - **FAISS: Replaced IndexHNSWFlat with IndexFlatIP** - HNSW was causing slow index builds (minutes) and Python crashes for 10K+ texts. IndexFlatIP builds instantly and matches the battle-tested pattern from TFM FULL, XLSTransfer, and KR Similar monoliths - **Batch encoding:** `model.encode()` now processes 100 texts per batch instead of all texts at once, preventing memory spikes - **Terminal cleanup:** Added `show_progress_bar=False` to all 7 `model.encode()` calls, eliminating tqdm "Batches: 100% 1/1" spam

Documentation: - New `docs/FAISS_IMPLEMENTATION.md` - Complete FAISS technical reference with monolith comparison - Updated User Guide with Quadruple Fallback (Fuzzy KR Match) section and FAISS troubleshooting

Technical Details: - `core/fuzzy_matching.py`: IndexFlatIP + batch_size=100 + show_progress_bar=False - `core/xml_transfer.py`: show_progress_bar=False on 2 encode calls - `config.py`: Removed HNSW params (FAISS_HNSW_M, EF_CONSTRUCTION, EF_SEARCH) - See `docs/FAISS_IMPLEMENTATION.md` for full technical documentation

Version 3.5.0 (February 2026)

New Features: - **Find Missing Translations** button in Quick Actions: Identify Korean strings in TARGET that are MISSING from SOURCE by (StrOrigin, StringId) key - Generates per-language XML concat files (`MISSING_FRE_*.xml`, `MISSING_GER_*.xml`, etc.) - Generates Excel summary report with per-language breakdown, Korean character counts, and word counts - Detail sheets in Excel with first 1000 entries per language - Optional export path filtering (System/Gimmick, System/MultiChange) - Word count statistics: Reports now include Korean character counts and word/sequence counts for translation effort estimation

Technical: - New `core/missing_translation_finder.py` module - MissingTranslationReport, LanguageMissingReport, and MissingEntry dataclasses for structured reporting

Version 3.4.0 (February 2026)

New Features: - Per-language failure XML files: When transfer fails, separate `FAILED_<LANG>_YYYYMMDD_HHMMSS.xml` files are generated per language with clean Loc-

Str format (no metadata, exact attribute preservation) - STRORIGIN_MISMATCH diagnostic: STRICT mode now shows specific "StrOrigin mismatch" error instead of generic "StringID not found" when StringID exists but StrOrigin differs - Transfer Scope option: Choose between "ALL" (transfer everything) and "Untranslated only" (skip entries with non-Korean target text)

Bug Fixes: - Fixed StringID matching bug: Added `.strip()` normalization to prevent whitespace-related match failures

Technical: - Failure XML files use `<root><LocStr ... /></root>` format for direct re-import - STRORIGIN_MISMATCH tracked separately in failure reports for better diagnostics

Version 3.1.0 (February 2026)

New Features: - Folder analysis on browse: detailed terminal output with file indexing, language identification, eligibility check - Cross-match analysis before TRANSFER: shows source-to-target file pairing - Window now vertically resizable (900×1000, min 900×900)

Bug Fixes: - Fixed TRANSFER button being invisible due to window too small (900×850) - Added error handling for folder analysis (permission errors, unreadable files)

Version 3.0.0 (February 2026)

New Features: - Added TRANSFER functionality - write corrections to XML files - Added transfer_file_to_file and transfer_folder_to_folder - Added transfer report with detailed statistics - Added mixed Excel/XML support in Folder mode - Added canonical text normalization (text_utils.py)

Improvements: - Unified normalization across all modules - Case-insensitive XML attribute reading - Better column detection in Excel files - Resource leak fixes in Excel operations - Improved error handling and logging

Technical: - Created core/text_utils.py as single source of truth - Created core/xml_transfer.py for transfer operations - Fixed newline order bug in XML parsing - Imported SCRIPT_CATEGORIES from config.py

Version 2.0.0 (February 2026)

New Features: - Added StringID-Only match type for SCRIPT strings - Added Strict match type (StringID + StrOrigin) - Added Special Key match type - Added Folder mode (recursive processing) - Added XML format input support - Added Reverse Lookup feature - Added branch selection (mainline/cd_lambda) - Added ToSubmit folder integration

Improvements: - Modular codebase (main.py + core/ + gui/ + utils/) - Better XML parsing with lxml recovery mode - Progress bar and detailed status updates

Version 1.0.0 (Initial Release)

- Basic substring matching
- Excel input/output
- StringID lookup
- Multi-language support (17 languages)

13.4 Support

Issues & Feedback: - GitHub Issues: LocalizationTools Repository

Documentation: - This User Guide: `docs/USER_GUIDE.md`

QuickTranslate | Version 3.7.0 | LocaNext Project *Lookup & Transfer - Two Tools in One*