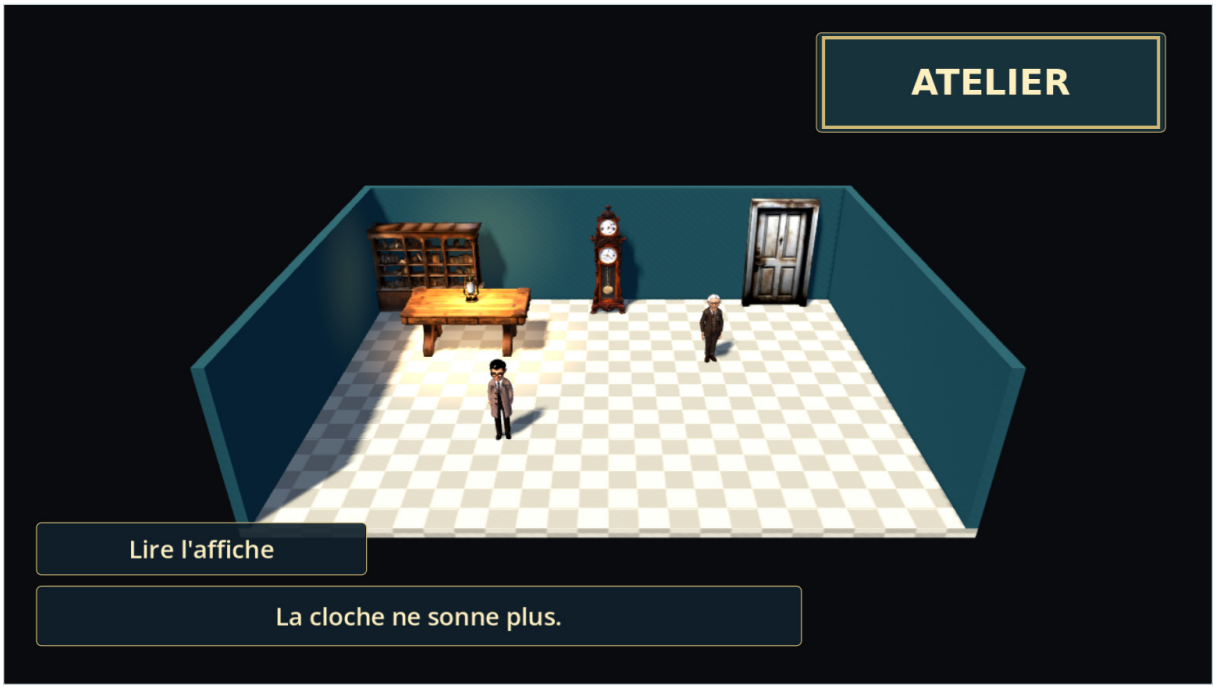
PCA 3D ADVENTURE STUDIO

# Game Translations

## Take your adventure into another language




### From the first sentence to the final language check

Add languages, translate dialogue and interfaces, assign images and voices, and share CSV files with a translation team. Step-by-step instructions, actual editor screenshots, and examples checked in the running game.

Version 0.86.3.4 - Windows, macOS and Linux

No practice project or extra downloads required.

# How to use this guide

Start with chapters 1-4 and get one sentence working in another language. Continue with the kinds of content your game uses. If someone else is translating for you, chapters 11-15 give you a complete export, return and review routine.

| Chapters | What you will do |
|---|---|
| 1-4 | Find Game Translations, add a language, save and test your first line. |
| 5-7 | Translate dialogue, GUI text, names, placeholders and scripted messages. |
| 8-10 | Assign replacement assets and voice recordings, understand movie subtitles and lip sync. |
| 11-15 | Export, edit and import CSV files, resolve conflicts, and divide work among translators. |
| 16-18 | Change languages through menus, action lists and scripts; check regional fallback, fonts and layout. |
| 19-20 | Test the game and its distributable build, then use the troubleshooting tables. |

The screenshots come from the running editor and game preview. Close-ups are cropped for readability. The workshop shows example names and IDs; use the corresponding records from your own adventure. The English-to-French example demonstrates the workflow rather than a finished translation of every item in the workshop.

The text, CSV previews, regional fallback, placeholders and GUI-image refresh were checked in this build. The voice screens show real bundled English audio assignments and an empty French voice field. You will need your own translated recordings and matching lip-sync work for a voiced release.

**A note about images in this version:** translated-asset lookup and already-loaded GUI textures are separate. Chapter 8 includes the tested action that refreshes a GUI image. The game screenshot on the cover includes that step.

You can follow the guide on Windows, macOS or Linux. The file picker may look different on your system, but the project workflow is the same.

# Contents

Click a chapter or page number to jump to it. PDF bookmarks include the smaller section headings.

1. What you are translating . . . . . 4
2. Open Game Translations and find your way around . . . . . 5
3. Add a language and choose a default . . . . . 8
4. Translate your first sentence . . . . . 10
5. Translate the rest of the adventure . . . . . 12
6. Keep source text and placeholders safe . . . . . 14
7. Translate text used by scripts and visual nodes . . . . . 17
8. Translate images and other assets . . . . . 18
9. Translate voice recordings . . . . . 21
10. Voice language, movie subtitles and lip sync . . . . . 24
11. Export a CSV for a translator . . . . . 26
12. Edit the spreadsheet without breaking it . . . . . 27
13. Import, review and save the returned work . . . . . 29
14. Deal with conflicts and rejected files . . . . . 33
15. Let several people work on translations . . . . . 36
16. Let the player change language . . . . . 38
17. Change language from scripts and visual scripting . . . . . 41
18. Regional languages, fonts and layout . . . . . 42
19. Test the translated game before release . . . . . 45
20. Quick fixes and a final checklist . . . . . 46

# 1. What you are translating

A translation lets the same game show different words, play a different voice recording, or load a different image according to the player's language. You keep one adventure: the rooms, puzzles and action lists stay together.

This guide starts with one English sentence, adds French, and follows that translation into the running game. After that, you will hand text to a translator in a CSV file, bring their work back, and deal with the problems that commonly turn up along the way.

The small workshop in the screenshots is an example, not a project you need to download. Follow the steps with a short message or GUI label in your own adventure. Nothing here requires a separate practice file.

## Three different language choices

| Choice | What it changes |
|---|---|
| Editor language | The words on the development tools and editor controls. This is an editor preference. |
| Game Language | The game's translated text and the language used when looking up translated assets. |
| Voice Language | Which available voice recording is requested. It can follow Game Language or use a separate language. |

There is also a subtitle-language setting for language-tagged movie subtitle cues. It is not a replacement for Game Language when translating ordinary dialogue. Chapter 10 explains the distinction.

Changing the editor to French does not translate your story. Likewise, you can develop in an English editor and release a game with several languages.

## What the translation system can hold

The catalogue gathers authored text from areas such as dialogue, GUI elements, object names, descriptions, quests and quoted script strings. It also includes common built-in game-menu labels. The asset side stores replacements for imported resources, such as another version of a sign image or an audio recording.

It does not write the translated words for you, redraw lettering inside an image, or record another actor's voice. Those are the materials you supply. Nor does a catalogue entry guarantee that every custom piece of code displays that string through the translation system. Test the place where the player actually sees it.

> **Before you continue**
> Start with one visible line. Prove that it changes in the running game before handing the whole project to several translators.

# 2. Open Game Translations and find your way around

1. Open your game in PCA 3D.
2. Open the **Project Settings** workspace. The page itself is headed **Project** followed by the game title.
3. Click **Game Languages & Translations...** above the project fields.
4. Resize the Game Translations window if you need more room. Scroll the right-hand panel to reach fields lower down.

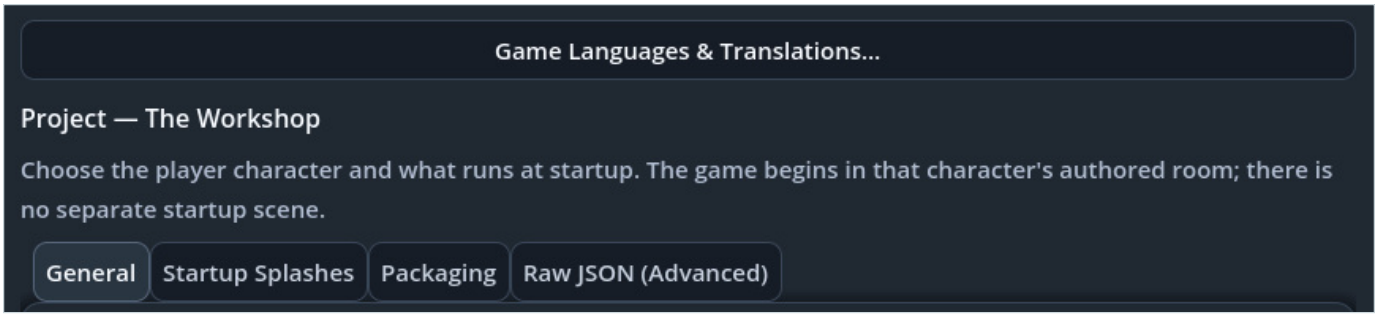



**Figure 1.** The translation button is above the Project form. You do not need to open the engine's own Godot project settings.

## Read the window from top to bottom

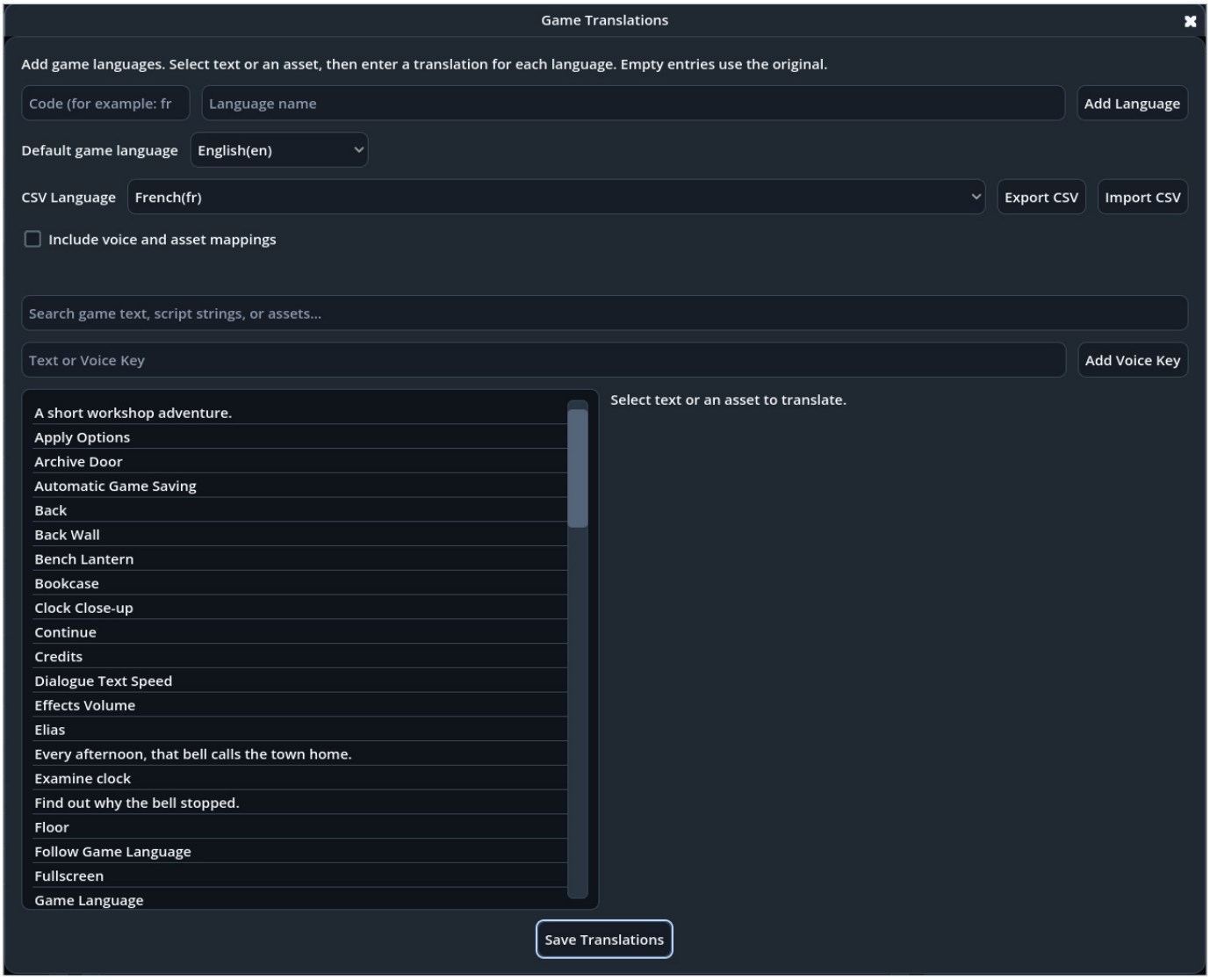



**Figure 2.** Game Translations after French has been added. The language setup and CSV controls are above the searchable catalogue.

| Area | Use it for |
|---|---|
| Code, Language name, Add Language | Register another language in this game. |
| Default game language | Choose the authored starting language for a fresh player. |
| CSV Language | Choose the language you are exporting or importing. |
| Export CSV / Import CSV | Send translations out to a spreadsheet and bring them back. |
| Include voice and asset mappings | Include resource IDs as well as written text in an export. |
| Search box | Narrow the list to words or asset names you recognise. |

| Area | Use it for |
| --- | --- |
| Text or Voice Key / Add Voice Key | Add a named entry for a voice mapping. This is optional. |
| Catalogue on the left | Select the original text or source asset. |
| Fields on the right | Enter text or choose resources for each language. |
| Save Translations | Put the window's current work into the open project document. |

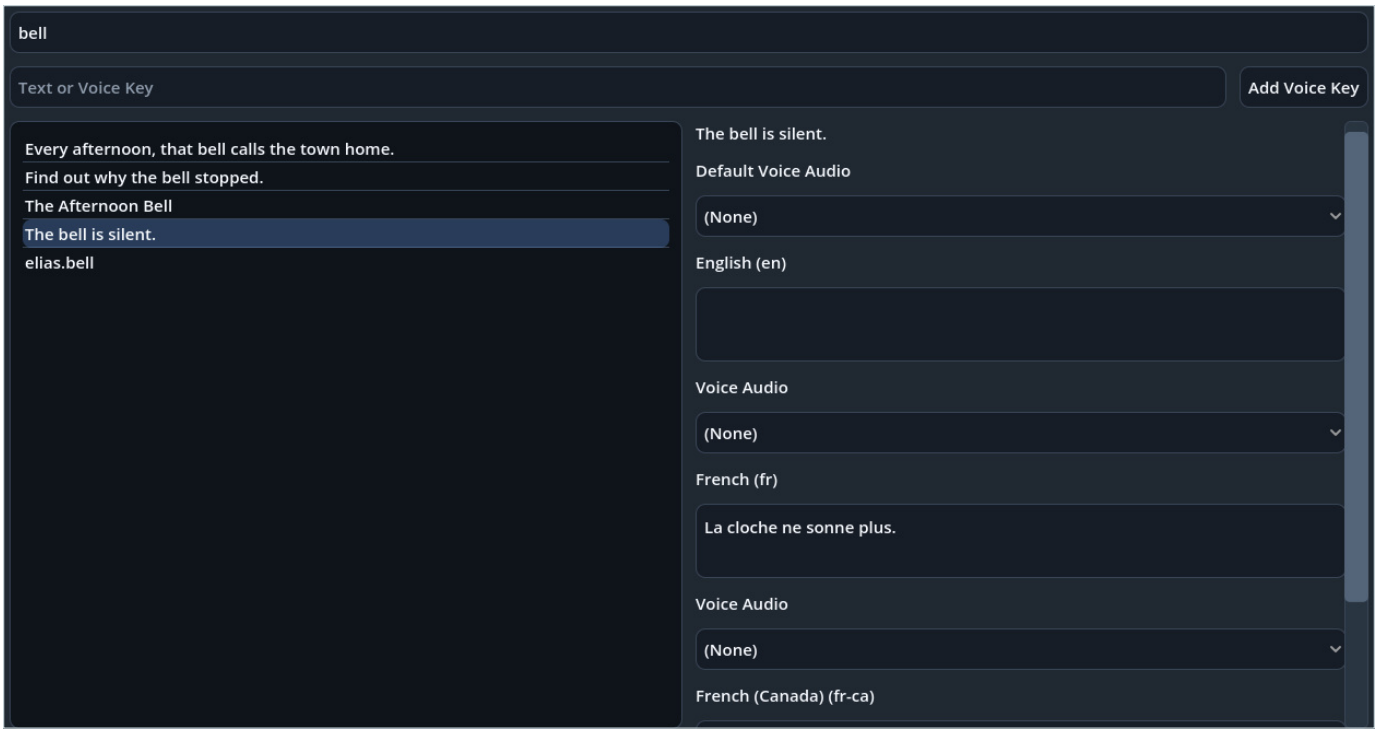



**Figure 3.** Searching for bell narrows the catalogue without changing the entries. Select the exact row you need.

Search is a filter, not a command to add a translation. If you type a phrase in Search, select the matching row beneath it before editing the right-hand fields.

There may be internal-looking words in the catalogue, particularly from scripts. Translate words players see. Leave command names, paths and record IDs alone.

## Understand the two normal saves

**Save Translations** accepts the work in this window. **Save Project** writes the whole adventure to disk. Use both.

1. Finish the translation edits.
2. Click **Save Translations**.
3. Click **Save Project** on the main toolbar.
4. Reopen Game Translations if you want to check the saved entries.

After a CSV import there is one earlier step: **Apply Changes** in the import preview. That accepts the reviewed rows into this window. It still needs the two saves above.

Do not rely on closing a window with its X as a way to save. If you have made changes you intend to keep, use the named save button first.

# 3. Add a language and choose a default

A language has a **code** and a **name**. The code is the stable identifier used by CSV files and scripts. The name is the readable label shown in the editor and language selectors.

## Add French

1. In **Code**, enter `fr`.
2. In **Language name**, enter `French` or `Français`, whichever label you want your players to see.
3. Click **Add Language**.
4. Check that the language now appears in **Default game language** and **CSV Language**.
5. Save Translations, then Save Project.




**Figure 4.** French(fr) is registered. CSV Language is French while the starting game language is still English.

Adding a language does not fill its text boxes. Empty boxes are allowed, which makes it possible to translate the game a section at a time.

## Pick sensible codes

| Language | Example code |
|---|---|
| English | `en` |
| French | `fr` |
| Spanish | `es` |
| German | `de` |
| Italian | `it` |
| Brazilian Portuguese | `pt-br` |

| Language | Example code |
|---|---|
| Canadian French | `fr-ca` |

The editor accepts a two- or three-letter base code, with optional hyphen-separated parts. It converts uppercase to lowercase and underscores to hyphens. For example, `PT_BR` becomes `pt-br`.

Use the same code throughout the game, returned CSVs and action lists. A second name with an existing code does not create another language. This window has no language-removal button in this version; decide on the codes before substantial translation work begins.

## Choose the starting language

**Default game language** is separate from **CSV Language**. Exporting a French CSV should not accidentally make French the starting language of your release.

Choose French as the default while testing the French edition, save both levels, then start a fresh preview. Before release, return the default to the language you want new players to receive.

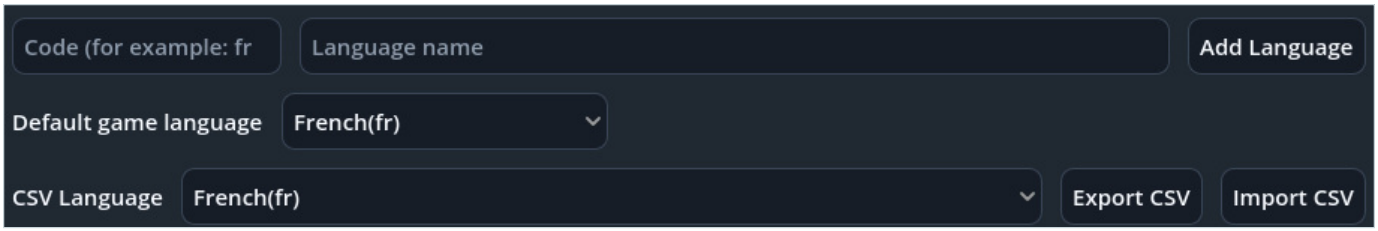



**Figure 5.** French is selected as the authored default for a test. It takes effect when Save Translations is used.

A returning standalone player may already have a saved language preference. That preference can override the authored starting value. Chapter 15 shows how to check both cases.

# 4. Translate your first sentence

The example source line is **The bell is silent.** It appears in a GUI label and in an authored dialogue line. Use any short, visible sentence in your own game.

1. Keep the original sentence in the dialogue, GUI or message action where you wrote it.
2. Open Game Translations.
3. Type part of that sentence in Search, such as `bell`.
4. Click the exact **The bell is silent.** row.
5. Find **French (fr)** on the right. Scroll that panel if necessary.
6. In the French text box, enter **La cloche est muette.**
7. Leave **Voice Audio** alone for now. Written translation does not require a recording.
8. Click Save Translations, then Save Project.




**Figure 6.** The original sentence is the entry heading. The French words belong in the French text box; the English box can stay empty.

The source remains English in the authoring field. The game uses the French translation when French is selected. Replacing the source field itself with French would change the lookup key rather than add another language.

## Check it in the running game

1. Set **Default game language** to French in Game Translations.
2. Save Translations and Save Project.
3. Stop an old preview, then use **Play / Restart** for a fresh run.
4. Reach the line or GUI element you translated.
5. Check the text as it appears in the actual interface.

A **Test Scene** is useful for reaching the room quickly. Use a full Play / Restart as well if the line belongs to the opening menu, startup script or cutscene.




**Figure 7.** French GUI text in the running adventure. The sign still says WORKSHOP here: an already-created GUI image needs its own refresh in this build, covered in chapter 8.

## What an empty translation does

With no French text for a source line, the game displays the original line. It does not automatically translate it, and ordinary text does not fall back through every language you have added.

To remove a translation, clear its text box in Game Translations and save. A blank cell in an imported CSV has a different purpose: it leaves the current translation untouched. Chapter 13 covers that difference.

> **Before you continue**
> Check the actual sentence, punctuation and accents. Seeing the correct language name in a selector is only the start of the test.

# 5. Translate the rest of the adventure

Once one sentence works, work through the game in a predictable order. A player usually meets the title/menu first, then the opening scene, then dialogue, inventory and journal entries.

## Dialogue, choices and messages

Search for the original spoken line, narrator message or choice caption. Translate each visible piece separately. A line and a choice that leads away from it are separate entries unless their source text is identical.

For the example notice, the useful entries are:

| Original | French example |
|---|---|
| The bell is silent. | La cloche ne sonne plus. |
| Read the notice | Lire l'affiche |
| Leave | Partir |

Some designers also offer a **Translate** button beside suitable text. That opens Game Translations with the relevant source selected. You can always return to the central window through Project Settings.

Read a whole conversation once before translating its pieces. A short choice such as "Leave" may mean leaving a room, abandoning an item or ending a conversation. The words around it decide the translation.

## GUI labels and built-in menus

Translate your own GUI captions, window titles, hints, empty-list messages and button text. Test their widths in the real GUI, including hover and pressed states when those states contain artwork.

The catalogue includes common built-in labels such as **New Game**, **Continue**, **Options**, **Save Game**, **Load Game**, **Game Language** and **Apply Options**. They need translations too if you use the built-in menus.

Adding French to the language list does not by itself translate those labels. A mixture of French dialogue and English menu labels usually means some entries remain blank.

## Names, descriptions, inventory and quests

Look up player-visible names and descriptions: the room name, an item's display name, quest title, objective text and journal entry. A proper name such as Ivo may stay unchanged, while "Workshop Key" probably needs a translation.

Keep **IDs** stable. For example, an item can have ID `workshop_key` and display name **Workshop Key**. Translate the display name, not the ID used by inventory actions and puzzle logic.

The catalogue can also contain names used only to organise your project. Seeing a name there is not a requirement to translate it. Check whether the player ever sees that name.

## When the same words appear twice

An identical source string has one shared translation. If two buttons both say **Back**, translating that entry affects both wherever supported text display uses it.

This is helpful until a word has two meanings. If **Open** means both opening a door and the adjective on a shop sign, use clear, distinct source wording such as **Open the door** and **Shop open**. Do not expect the translation window to distinguish two identical source strings by room or object.

# 6. Keep source text and placeholders safe

The source sentence is the translation's lookup key. Capital letters, punctuation and spaces matter.

| Authored text | Same entry as "The bell is silent."? |
|---|---|
| The bell is silent. | Yes. |
| The bell is silent! | No: punctuation changed. |
| the bell is silent. | No: the initial letter changed. |
| The bell is now silent. | No: the wording changed. |

When you revise a line, search for its new wording and translate it. An old translated key can remain in the catalogue even after you stop using that sentence. Its presence is not proof that the revised line is covered.

## Preserve the little pieces the game fills in

A placeholder is a marker the game replaces with a current value. For example, `@PlayerName` becomes the active player's name and `$$score` becomes the value of the `score` variable.

Source:

```
Hello, @PlayerName. Score: $$score
```

French translation:

```
Bonjour, @PlayerName. Score : $$score
```

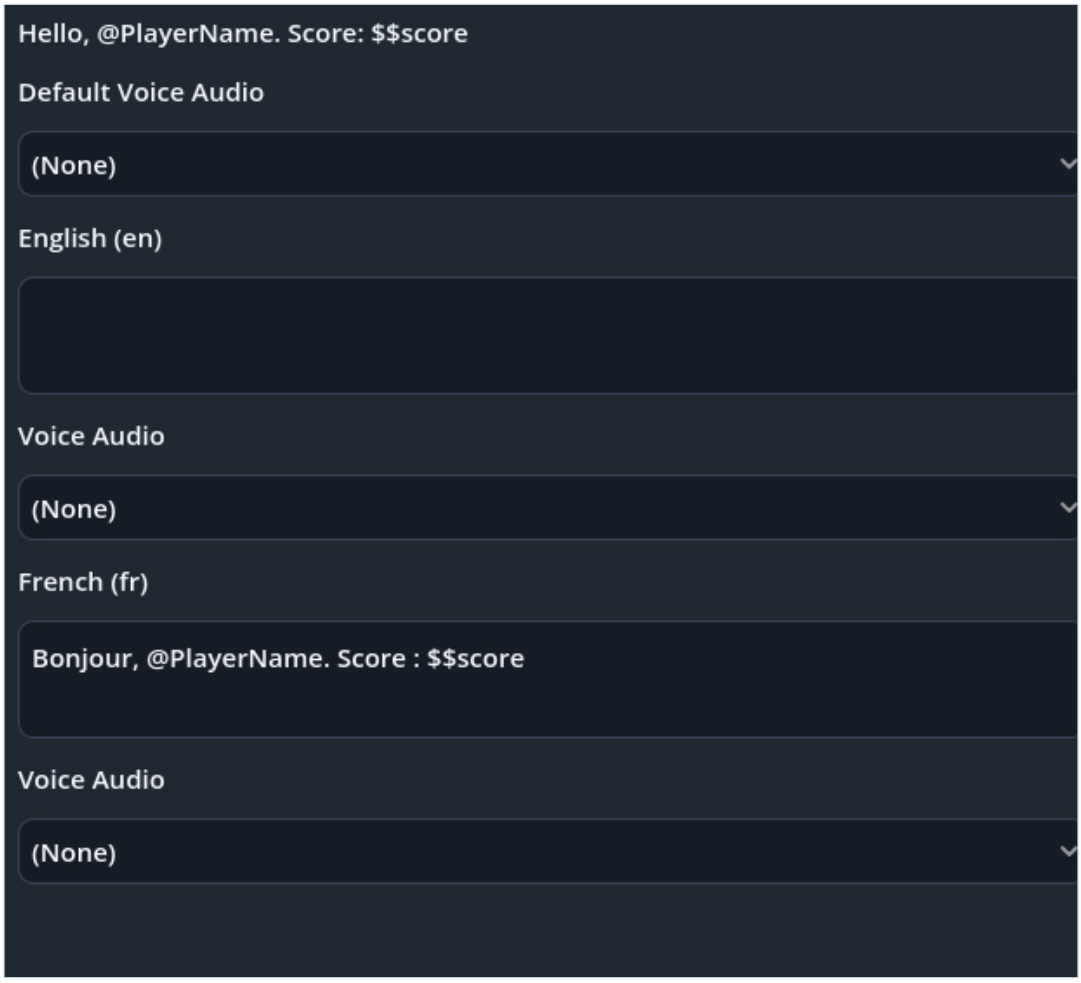



**Figure 8.** The French sentence keeps @PlayerName and $$score exactly as written. Only the surrounding words and spacing have changed.

Keep the marker's spelling, case and number of occurrences. You can move it to a natural place in the translated sentence, but do not replace it with a sample value such as Ivo or 7. That would freeze one example into the translation.

| Marker | Meaning |
|---|---|
| `@PlayerName` | Active player's display name. |
| `@PlayerId` | Active player character's ID. |
| `@SceneName` | Current scene's name. |
| `@Score` | The score value. |
| `$$variable_name` | The named game variable's value. |
| `@//@` | Separator used by random text alternatives. |
| `@` followed by two vertical bars and `@` | Separator used by sequential text alternatives. |

If an original uses alternative-text separators, keep the same separators in the translation and translate the alternatives on either side. The CSV check compares the supported markers, but it does not check whether the translated sentence makes sense.




**Figure 9.** A running message uses the French base translation for fr-ca and fills in Ivo and score 7.0. The markers remain in the authored text, not those example values.

## Keep formatting and meaning intact

Preserve any tags or escape sequences your particular text field needs. The built-in CSV checks are not a general proofreader for every custom markup format. Check deliberate line breaks, quotation marks, speaker names and formatting in the final display.

For puzzles, translate clues together with their answers. A wordplay puzzle can stop making sense even when every sentence is grammatically correct. Record the intended solution in the translator's notes.

# 7. Translate text used by scripts and visual nodes

A message written in an action list or visual node is still game text. Write the source sentence in the action's normal **Text** field, then translate the same sentence through Game Translations.

The catalogue also reads quoted strings from script source. It does not rewrite your GDScript, PCA Script, Lua or BASIC program. Translation happens when the original string reaches a supported game text-display function.

## A small scripted message

Inside the entry function already supplied by a GDScript template, this displays a source message:

```
await ShowMessage({
    "text": "The bell is silent.",
    "duration": 2.0
})
```

Keep the template's existing entry function and parameters. There is no need to create another script class just to translate one message.

With French selected and a French entry for that exact source sentence, the displayed message uses the French words. The same source sentence in a visual message action uses the same translation.

## Write a whole sentence when you can

Prefer a stable source such as **Hello, @PlayerName.** to building a new string from separate translated fragments. The translation lookup needs to recognise the complete source string.

If a script generates **Hello, Ivo.** by joining strings, it may no longer match the catalogue entry **Hello, @PlayerName.** A translator also needs freedom to change word order. A complete sentence with placeholders gives them that freedom.

Quoted strings such as `workshop_key`, `duration` or a command name can appear in the script catalogue. Leave them alone unless they are deliberately shown to the player. Translating code vocabulary can break the script without improving the game text.

> **Before you continue**
> Run the message through the same action or script the player will use. A translated catalogue row alone cannot prove that a custom display path uses it.

# 8. Translate images and other assets

An asset translation is a replacement resource. For an English sign image, the French entry points to an imported French sign image. The system is choosing between files you provide.

## Prepare the two versions

1. Keep the original image in your image editor.
2. Make a translated version with the same intended proportions and transparent areas.
3. Import both into PCA 3D's asset library as images.
4. Give them clear names, for example **Workshop Sign - English** and **Workshop Sign - French**.
5. Keep their IDs distinct, such as `sign_en` and `sign_fr`.

Use the original asset in the scene or GUI. The translation mapping belongs to that original asset's entry.

## Assign the replacement

1. Open Game Translations.
2. Search for the original asset's name.
3. Select its **[Asset: image]** entry.
4. Under French, choose **Workshop Sign - French(sign_fr)**.
5. Leave English on **Use original asset**.
6. Save Translations and Save Project.

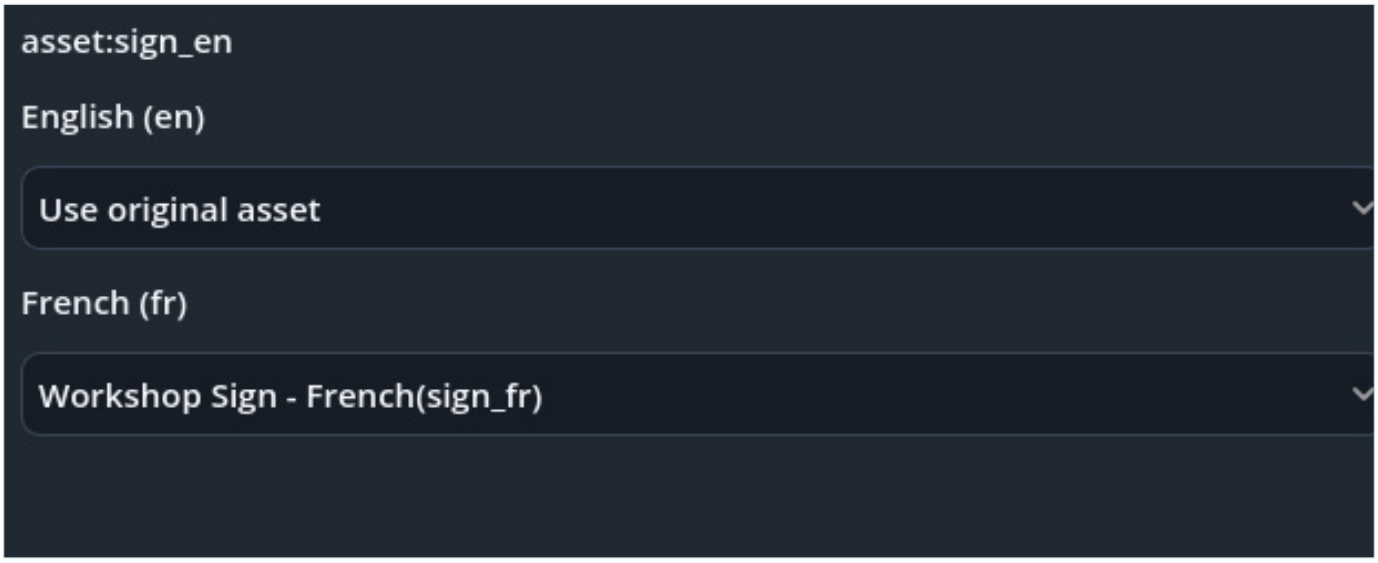



**Figure 10.** The selected source is asset:sign_en. French points to the imported French image while English uses the original.

The picker offers assets of the same kind. An image needs an image, a model needs a model, and so on. A matching kind is only the first check: a replacement character model still needs compatible animation, skeleton and attachment setup.

This approach can also be used for suitable imported audio, video, fonts or other resources where the game's loading path uses the translated asset lookup. Always test the specific use. A CSV mapping contains IDs, not the actual image, font or recording.

## Refresh a GUI image after a language change

In this version, an image control can keep the texture it loaded earlier. In the workshop test, the French text appeared while the sign still showed WORKSHOP, including after starting a fresh preview. The mapping itself correctly resolved to the French asset, but that did not rebuild the existing image control.

For the tested GUI image, reapplying its original asset ID through **ChangeGUIElementProperty** refreshed the image using the current language:

| Action | Values |
|---|---|
| Set Game Language | Language Code `fr`; Save Player Preference as appropriate. |
| ChangeGUIElementProperty | GUI `translation_hud`; Element `workshop_sign`; Property `assetId`; To `sign_en`. |

Keep `sign_en` in that second action. It is the source ID, so the French mapping can select `sign_fr`. Hard-wiring `sign_fr` would bypass the useful source-to-language relationship.




**Figure 11.** After the image property is reapplied, the running game shows ATELIER and the French text. This screenshot includes that refresh step.

Place the refresh after the language-setting action in your language-selection routine. If you need it at startup, run it after the GUI exists. Hiding and showing a GUI is not a reliable way to force a texture reload; it may simply reuse the same control.

For another image, video, font or model, test that resource's own loading path. Do not assume the GUI-image workaround rebuilds every resource type. Keep visible lettering in normal GUI text where possible; it is easier to translate and maintain than text painted into artwork.

# 9. Translate voice recordings

Written text and speech recordings are separate. A French sentence can be displayed while the available English recording plays. That is a useful fallback during development, but it is not a French voice edition.

The workshop example contains bundled English recordings. It does not contain a French actor's take, so the screenshots deliberately leave the French **Voice Audio** field empty.

## Assign a voice to a source sentence

1. Import the finished recording as an audio asset.
2. Open Game Translations and select the original sentence.
3. Choose the usual recording in **Default Voice Audio**, if you want one.
4. Under a particular language, choose its matching recording in **Voice Audio**.
5. Save both levels and play the actual line.




**Figure 12.** The English bell line has a real English recording in Default Voice Audio. Its French text is entered, but French Voice Audio is still (None).

Default Voice Audio is an audio fallback. It is not the same field as Default game language.

Use names that identify the speaker, line and language. For example, **Elias - Bell - French** is much easier to audit than **voice17_final_final**. In CSV mappings, use the actual asset ID rather than guessing from its filename.

## Understand the two assignment routes

| How the line is authored | Where its translated voice is looked up |
|---|---|
| No explicit Voice Audio, no Voice Key | Voice mappings under the original source text. |
| A Voice Key is supplied | Voice mappings under that key. |
| An explicit Voice Audio asset is supplied | Language replacements for that source audio asset. |

An explicit Voice Audio assignment takes the asset route. Do not expect a different mapping under the sentence or Voice Key to override it automatically. Choose a consistent route for a conversation and test it.

## Use a Voice Key when stable labels help

A Voice Key is a short label for a voice mapping, such as `elias.bell`. It is useful when you want the recording's reference to remain stable while you revise the displayed sentence.

1. Enter `elias.bell` in **Text or Voice Key**.
2. Click **Add Voice Key**.
3. Choose its Default Voice Audio and any language recordings.
4. In the speaking action or dialogue field that supports it, enter that exact **Voice Key**.
5. Keep the displayed source sentence in the normal Text field.

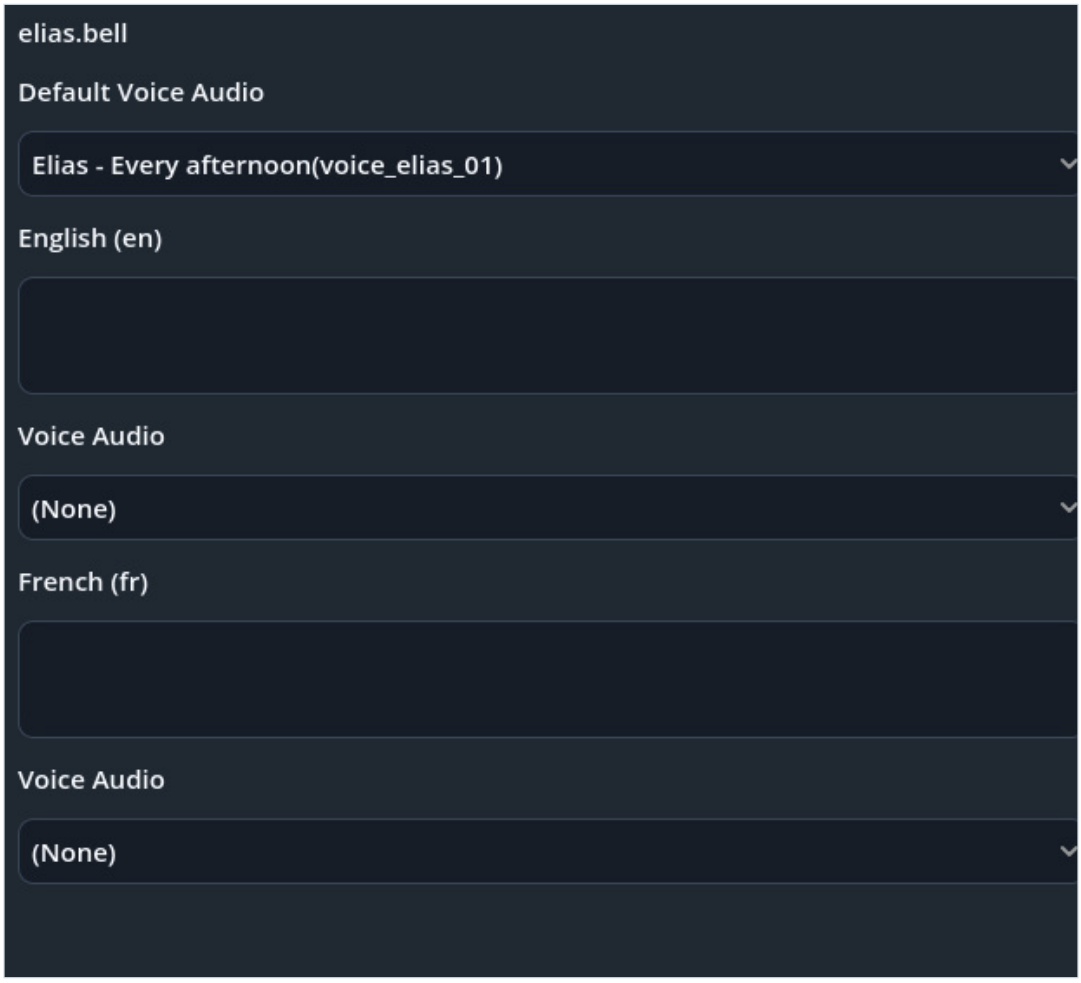



**Figure 13.** elias.bell is a named voice entry. Adding it does not automatically assign it to every bell-related line.

A Voice Key is not an automatic dialogue node or a replacement for the sentence players read. Adding the key creates a mapping entry; the speaking action must actually use it.

## What happens when a recording is missing

For voice mappings, the system can try the requested language, its base language, the project's default language, and the Default Voice Audio mapping. An explicitly assigned source audio asset can be the final fallback on the asset route.

For example, `fr-ca` can use an available `fr` recording. With no French recording, an English default may be selected. If no usable candidate exists, there is no translated voice to play. Reading the French text is still possible.

Listen to the complete line. Check that it is the right speaker, take and language, starts at the right time, and is not much louder than adjacent recordings. A valid asset ID cannot establish any of those things.

# 10. Voice language, movie subtitles and lip sync

## Give players a voice-language choice

The built-in game Options menu provides **Game Language** and **Voice Language**. Voice Language can use **Follow Game Language**, so selecting French text also requests French speech where available.

A player may prefer French text with English speech. That is a separate voice-language choice, not another set of French text translations.




**Figure 14.** The running game's Options menu has separate Game Language and Voice Language controls. Menu captions still need their own translations.

When using that menu, choose the values and click **Apply Options**. The built-in menu saves the player's options. Custom language buttons should deliberately decide whether to save the preference; chapter 16 shows the action.

## Movie subtitles are another layer

Language-tagged movie subtitle cues use **ChangeSubtitleLanguage**. Each cue has a language code, text and timing. A cue tagged `fr` is selected with the French subtitle language.

Ordinary dialogue and GUI text still use Game Language. Changing only the movie subtitle language will not turn the whole game's captions French.

If you author movie cues in a data field or script, keep their start/end times and language tags intact. Translate the cue text, then check it against the finished movie. A translated video asset and a translated subtitle track are two different resources.

## Match lip sync to the translated recording

A different recording has different mouth timings. Reusing the English lip-sync timing with a French take can make the character speak after the mouth has stopped, or close its mouth midway through a word.

1. Finish the translated recording first.
2. Use the character and lip-sync workflow to generate or import timings for that recording, using the built-in tools or a suitable Rhubarb track.
3. Check the character's mouth/viseme mapping and the portrait mapping used by that speaker.
4. Play the translated line with that recording selected.
5. Check both the in-scene character and its portrait, if your dialogue shows one.

The character's mouth-shape mapping can be reused when appropriate. The timing track must fit the take that is actually playing. This translation window chooses audio; it does not by itself generate a new lip-sync performance.

# 11. Export a CSV for a translator

A CSV is a plain text table that a spreadsheet program can open. It lets someone translate without working directly in your project. They return the table, and you review the proposed changes before accepting them.

## Make the handoff

1. Save your project, including any recent source-text changes.
2. Open Game Translations.
3. Set **CSV Language** to the language this person is translating, for example **French(fr)**.
4. Leave **Include voice and asset mappings** off for a text-only handoff.
5. Click **Export CSV**.
6. Choose a folder and a clear filename, such as `workshop-fr-text-01.csv`.
7. Keep an unchanged copy of this export.
8. Give the translator a working copy and brief notes about the game.

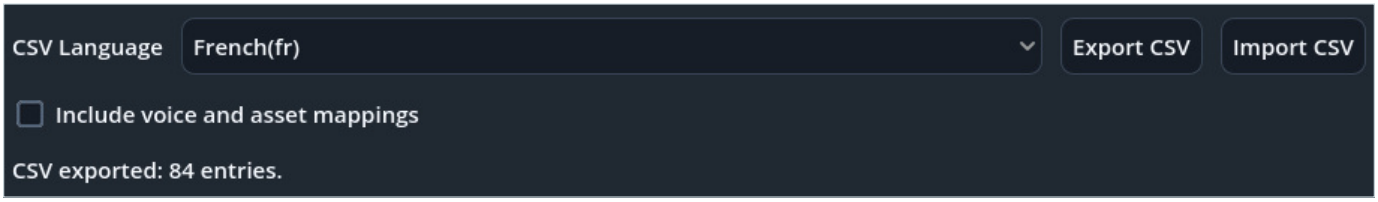



**Figure 15.** The editor confirms that the text CSV was exported. Check CSV Language before exporting each handoff.

Use a date or revision number in the filename, plus the language code. The filename helps people organise their work; the language stored inside the file is what the importer checks.

For written translation alone, the translator does not need your models, engine installation or whole project archive. They do need context: who is speaking, what a short button does, and whether a line is a clue or a joke.

## When to include resource mappings

Tick **Include voice and asset mappings** when a technical coordinator needs to assign imported audio or replacement resources in the table.




**Figure 16.** With mappings included, the export contains voice and asset assignment rows as well as text.

Those rows contain resource references. They do not bundle the recordings or images into the CSV. Import the replacement assets into the project separately and agree on their IDs before filling those rows.

Keep the text-only export for the person translating prose unless they are also responsible for the technical assignments. A long list of asset IDs is distracting when the job is simply to translate dialogue.

### CSV or TSV?

A `.csv` export uses commas between columns. A `.tsv` export uses tabs. Either can be useful; choose the one your spreadsheet handles reliably and preserve its text encoding.

The importer recognises comma, tab and semicolon separators from the header. Exported files use UTF-8, which preserves accented and non-English characters. The import size limit is 16 MiB per file. If a very large project exceeds it, divide the returned work into valid files with complete headers and non-overlapping rows.

## 12. Edit the spreadsheet without breaking it

Open the file using your spreadsheet program's text/CSV import option if necessary. Choose **UTF-8** and the separator that matches the file. The first row should become seven separate column headings.

### What the columns mean

| Column | Meaning and handling |
|---|---|
| `format` | File-format marker. Keep `pca3d-game-translations-v1` unchanged. |
| `language` | The target code, such as `fr`. Keep it unchanged. |
| `type` | `text`, `voice` or `asset`. Keep it unchanged. |
| `key` | The exact source-text key or resource/key reference. Keep it unchanged. |
| `source` | The original text or source label provided by the export. Keep it unchanged. |
| `translation` | The column the translator normally edits. |
| `original` | The translation that existed when the file was exported. Keep it unchanged; it is used to detect conflicting edits. |

The two columns called **source** and **original** are easy to confuse. Source is the original game wording or exported source label. Original is the older translation baseline, which may be empty on the first handoff.

## A small example

This shortened table shows the meaningful text columns. The actual CSV still needs all seven columns.

| source | translation | original |
|---|---|---|
| The bell is silent. | La cloche ne sonne plus. | La cloche est muette. |
| Read the notice | Lire l'affiche | Empty in this first handoff. |
| Leave | Partir | Empty in this first handoff. |

The translator changes only **translation**. They leave the older **original** value in place even when they disagree with it. That lets the importer tell a proposed revision from an accidental overwrite of newer project work.

## A reliable spreadsheet routine

1. Keep the header row.
2. Display source and translation side by side.
3. Set reference/ID columns to text where your spreadsheet offers that option.
4. Translate in the **translation** column.
5. Preserve placeholders and intentional line breaks.
6. Filter rows if useful, but sort whole rows together, never just one column.
7. Save or export the result as **CSV UTF-8** or the agreed TSV format.
8. Accept the spreadsheet's text-format export warning only when you mean to save a CSV, not an Excel workbook.
9. Reopen the exported file briefly to check that the columns and accents survived.

Do not add a comments column to the import file. Keep translator questions in a separate notes document using the exact source line or key as the reference. Do not send an `.xlsx` file back with `.csv` added to its name; renaming a workbook does not convert it to plain text.

A comma, quotation mark or line break inside a translated sentence is allowed when the spreadsheet saves proper CSV quoting. Let the program handle that quoting. Hand-editing commas in a text editor is a good way to shift the rest of the row into the wrong columns.

## Leave unfinished cells blank

A blank translation means "no proposed update" during CSV import. It does not mean "delete the translation already in the project".

A translator can also return only their assigned rows, provided the required header and complete columns remain. Rows omitted from a valid return file are left alone. This is useful when dividing one language among several people.

> **Before you continue**
> Keep an untouched export and an untouched returned file. If something goes wrong, you can compare them without guessing which cells changed.

# 13. Import, review and save the returned work

Save a working copy of your project before merging returned translations. Then use the preview as a review screen, not as a formality.

## Import the example return

1. Open Game Translations.
2. Set **CSV Language** to **French(fr)**, matching the file.
3. Click **Import CSV** and choose the returned file.
4. Read the summary at the top of **Preview Translation Import**.
5. Compare **Current Translation** with **Imported Translation**.
6. Check the **Import Status** for every changed row.
7. If the proposed changes are correct, click **Apply Changes**.
8. Click **Save Translations** in the parent window.
9. Click **Save Project** on the toolbar.
10. Run the translated section of the game.

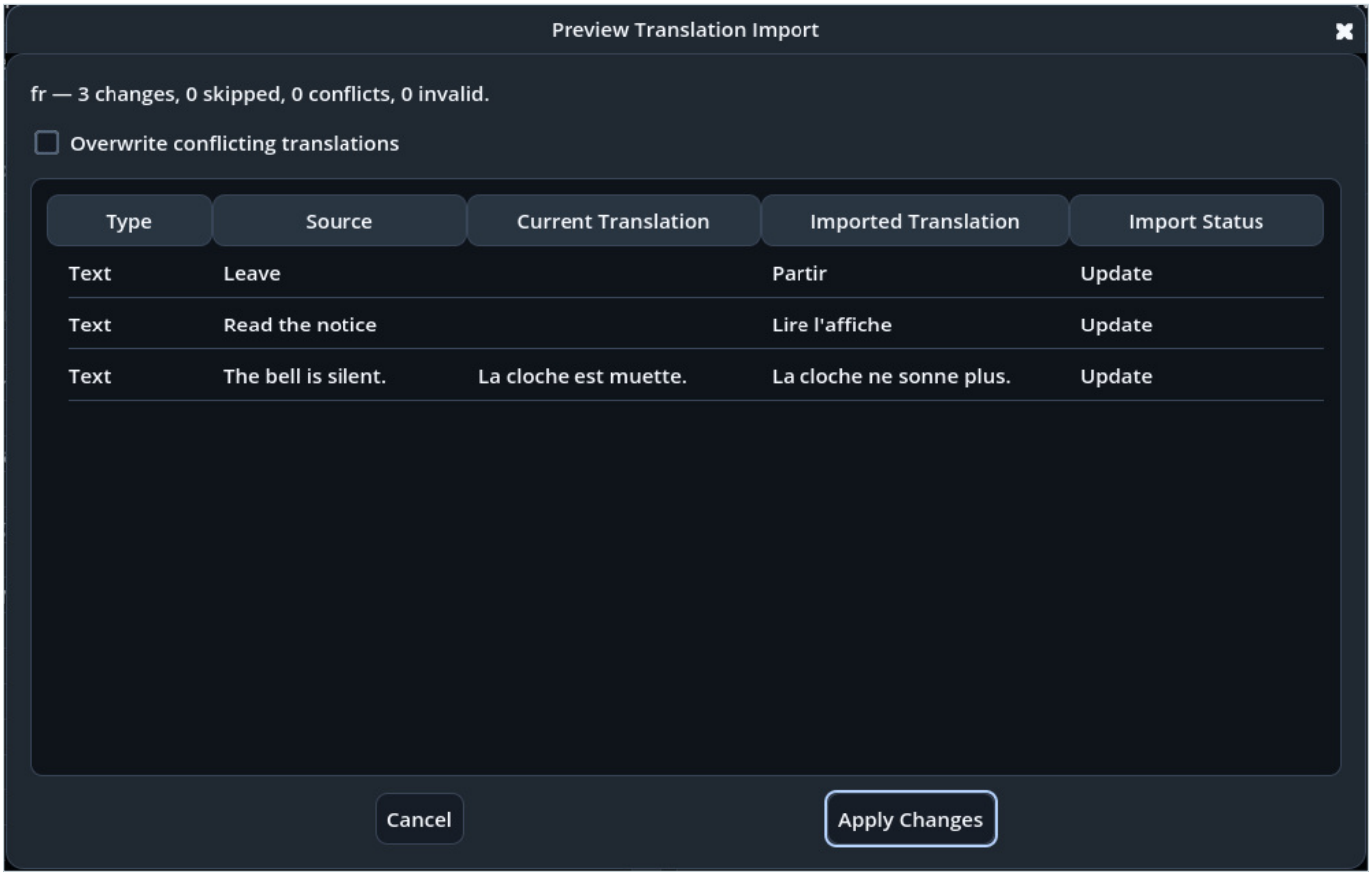



**Figure 17.** The example return proposes three updates. The current and incoming bell translations are visible side by side before anything is applied.

The summary separates changes, skipped rows, conflicts and invalid rows. "Skipped" does not always mean something is wrong. An unchanged or blank translation is deliberately skipped.




**Figure 18.** After Apply Changes, the status tells you to Save Translations and then save the project.

## Understand the row statuses

| Status | What happens |
|---|---|
| Update | The proposed translation can be applied. |
| Unchanged | No useful new proposal; the current project value is left alone. |
| Blank translation skipped. | The incoming cell is empty; the current value is left alone. |
| Conflict | The project changed since export. It is protected unless you deliberately allow overwrite. |

| Status | What happens |
| --- | --- |
| Overwrite conflict | You have enabled the overwrite checkbox for a conflicting proposal. |
| Unknown or changed CSV entry. | The source/key information does not match an accepted current entry. |
| Translation placeholders do not match. | Required tokens differ; this row is not applied. |
| Invalid translation asset. | The resource reference is missing or the wrong kind. |

Valid Update rows can be accepted even when other rows are invalid or protected conflicts. Read the counts so you know what was left behind. Repair rejected rows and import a corrected file afterward.

The Apply Changes button is disabled when there is nothing eligible to apply. A file with only unchanged rows is not a failed translation; it simply proposes no change.

## Why an old return may be marked Unchanged

Suppose the export contained a French translation, you improve that translation inside the editor, and the translator returns the old value without changing it. The importer can recognise that the return equals its exported baseline and leave your newer project wording alone.

That protection depends on keeping **original** unchanged. Editing the baseline makes the comparison unreliable.

## Clear a translation in the editor

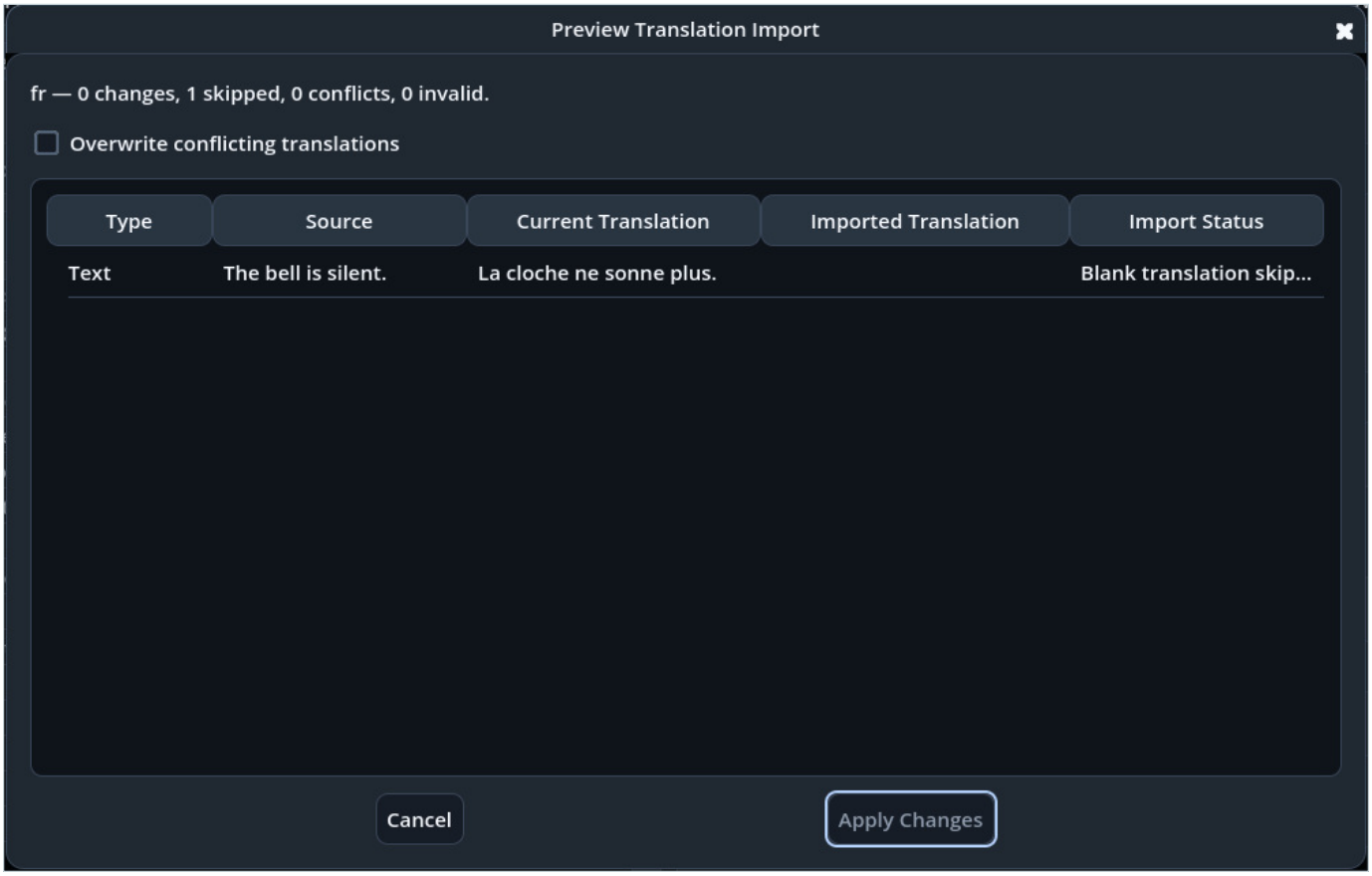



**Figure 19.** An empty incoming cell is skipped. Notice that the current bell translation is still present.

To remove the French wording deliberately, cancel the preview, select the source in Game Translations, clear the French text box, then Save Translations and Save Project. For a resource mapping, choose **Use original asset**; for a voice mapping, choose **(None)** at the relevant level.

The absence of a language-specific voice mapping may still allow another voice fallback. Removing one mapping does not necessarily mean silence.

# 14. Deal with conflicts and rejected files

## A conflict means two versions changed

Here is a typical case:

1. You export a row whose French translation is **La cloche ne sonne plus.**
2. Someone changes that row inside the project to **La cloche ne sonne plus !**
3. The translator returns a different proposal: **La cloche est silencieuse.**
4. The importer sees that both the project and the return differ from the export baseline.

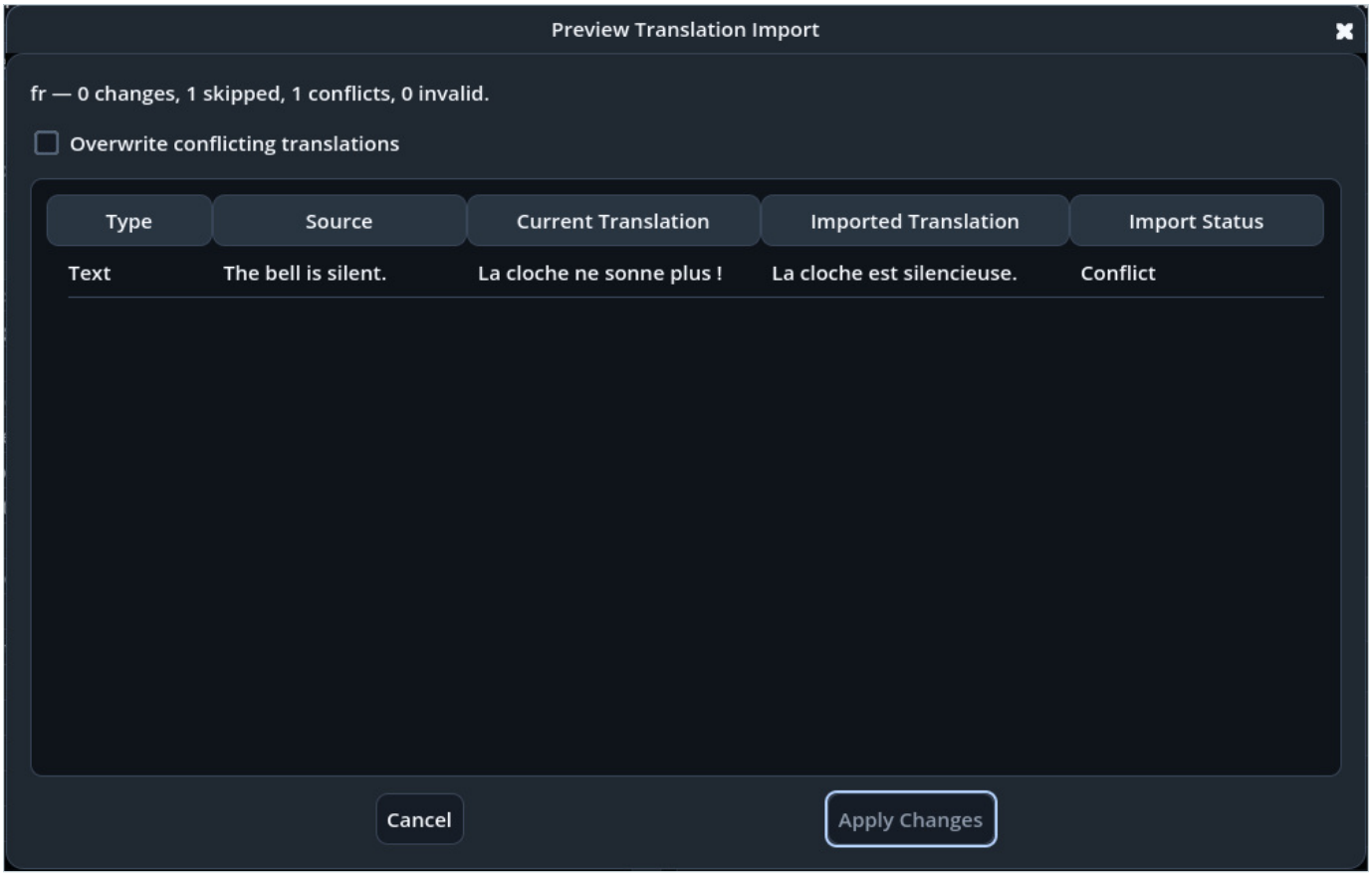



**Figure 20.** The conflicting row is protected. With no other valid updates, there are zero changes available to apply.

Decide which wording to keep by comparing the meanings and punctuation. The computer cannot judge which sentence is better.

If the current project wording wins, cancel or leave the conflict protected. If the returned wording wins, enable **Overwrite conflicting translations**, check the rows again, then apply and save both levels.

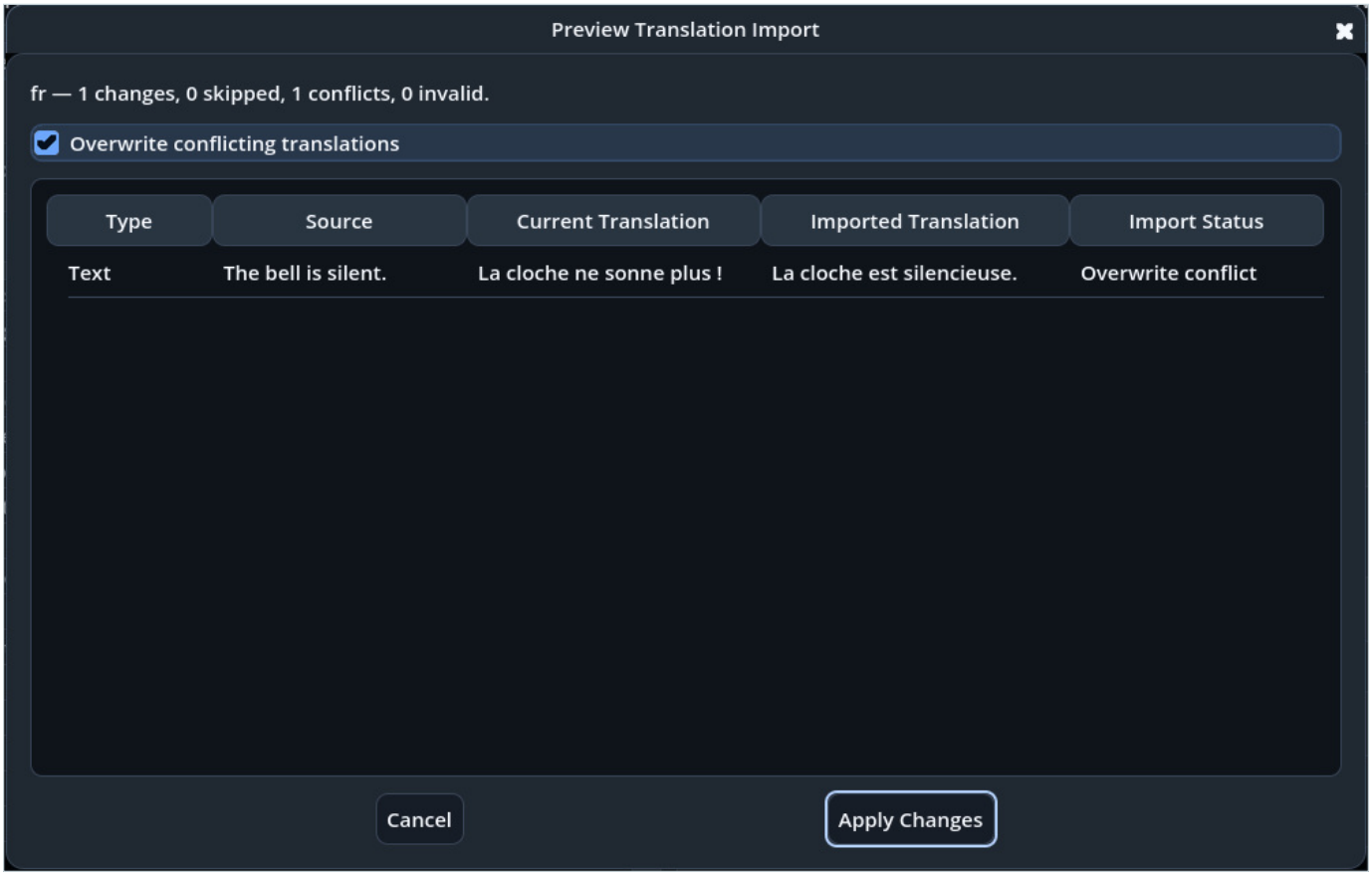



**Figure 21.** Enabling overwrite makes the proposed conflict eligible. The conflict count remains visible as a reminder that you are replacing project work.

The checkbox applies to the conflicting rows in that preview. For different decisions on several rows, prepare a smaller return containing only the approved proposals, or edit the chosen entries directly. Avoid approving an entire batch merely to accept one good sentence.

## Missing placeholders and invalid resources

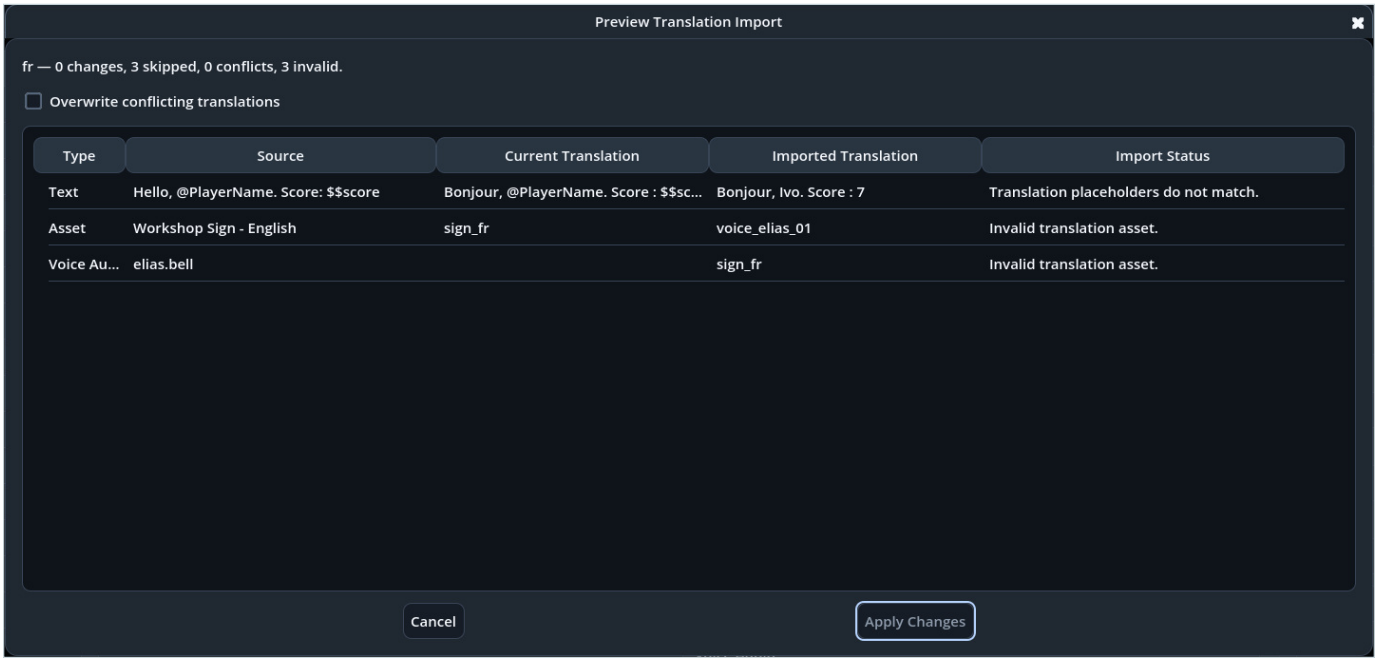



**Figure 22.** This deliberately incorrect return replaces placeholders with sample values and assigns the wrong resource kinds. All three rows are rejected.

For a placeholder problem, compare the source and translated markers character by character. Restore `@PlayerName` or `$$score` rather than entering the values you saw during a test.

For an asset problem, check that the target is imported into this exact project and that its ID is correct. An image ID cannot be used as a voice recording. An audio ID cannot replace an image. Renaming a file on disk does not automatically change the asset ID stored in the project.

Some mistakes reject the whole file before a usable preview appears:

| Message | What to check |
|---|---|
| CSV language does not match the selected language. | The CSV Language selector and every row's language code must agree. |
| Duplicate CSV entry. | The same type/key pair appears more than once. Remove the accidental duplicate. |
| Invalid CSV file. | Check encoding, required headers, format marker, column structure and that the file is really CSV/TSV. |

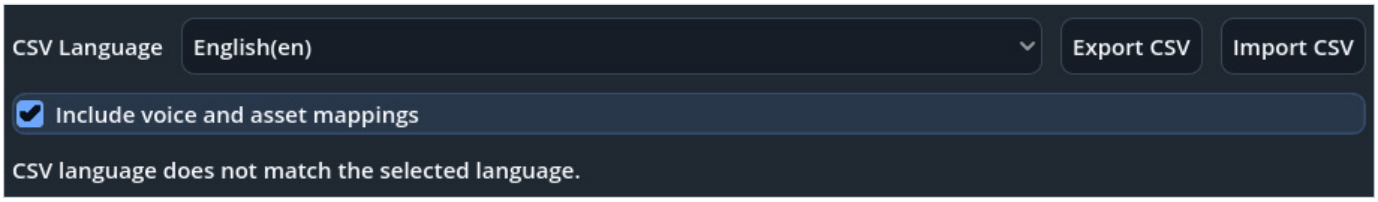



**Figure 23.** The selected language is English but the returned file contains French rows. The editor rejects the mismatch.




**Figure 24.** The same type/key pair appears twice in this test file. It is rejected rather than silently choosing one translation.

A text row and a voice row can legitimately share a key because their types differ. Two text rows with the same key are a duplicate.

If the editor reports **Translations changed. Preview the CSV again.**, the values changed after the preview was prepared. Reimport or refresh the preview, compare it again and then apply. Do not keep approving an old comparison.

# 15. Let several people work on translations

The simplest arrangement is one person per language and one person responsible for merging the returns into the master project.

## Agree on the basics before exporting

Give each translator the target language code, the game's tone, a small glossary, the names that must remain unchanged, and the expected file format. Explain placeholders and tell them to edit only the translation column.

For a workshop adventure, the glossary might distinguish **bell**, **chime**, **clock tower**, **workshop** and **study**. Agree on how characters address each other, too. Those choices are harder to repair after five people use five different terms.

A separate notes sheet can have these columns: source/key, speaker or screen, context, question, agreed answer. Keep it outside the import CSV.

## One language, several translators

Divide the rows into non-overlapping assignments: for example, menus and GUI, opening chapter, later rooms, and journal/inventory. Each returned file needs the complete CSV header and the original seven columns, but it can contain only that person's rows.

Remember that identical source wording is shared. A line such as **Back** belongs to one key even if it appears in several rooms. Assign that shared row once, or have the coordinator resolve it before merging.

## A practical handoff schedule

1. The coordinator saves a named project revision and exports a language CSV.
2. The coordinator keeps the original export and divides working copies by agreed rows.
3. Each translator works only on their assigned rows and keeps the baseline column intact.
4. Returns get distinct filenames, such as `workshop-fr-menus-return-01.csv`.
5. The coordinator imports one return at a time and reviews its preview.
6. After each accepted batch, the coordinator saves translations and the project.
7. A language reviewer reads the combined text for consistent terms and tone.
8. Someone plays the translated game and records layout, voice and puzzle issues.
9. A fresh export carries the accepted work into the next revision round.

Do not merge separate copies of the whole project just to combine translations. CSV handoffs are easier to review and avoid unrelated scene or logic edits being mixed into language work.

## When the author changes the story

Tell translators which source revision they received. If you rewrite a sentence midway through translation, send the new wording and context; do not assume the old row will follow it automatically.

The importer catches mismatched source/key data in a returned row. However, old translation keys can remain in the catalogue. A return for an old sentence may still be a valid entry while your newly rewritten sentence is untranslated. Search for the new sentence and test it in the game.

Do not treat catalogue size as a progress percentage. The list may include unused old wording, technical script strings and internal names. Keep a list of player-facing screens and conversations, then check those sections in play.

# 16. Let the player change language

There are two common routes: the built-in Options menu, or your own GUI language buttons.

## Use the built-in Options menu

1. Register the languages in Game Translations.
2. Translate the built-in menu labels you use.
3. Run the game and open its menu, then **Options**.
4. Choose **Game Language**.
5. Choose **Voice Language**, or **Follow Game Language**.
6. Click **Apply Options**.
7. Return to the game and check text and resources.

The built-in Apply Options saves player preferences. In a standalone build, those saved choices can be loaded on the next run. The embedded editor preview deliberately starts from authored defaults rather than loading the standalone player's existing preference file.

This is why an editor test and an old installed build can begin in different languages without a translation being lost.

## Make a French button in a GUI

1. Add a Button element to your language-selection GUI.
2. Give it a clear label, such as **Français**.
3. Select the button and open its **Events** tab.
4. Find **Click**, expand **Additional Event Actions**, and use **Add Next Action** to add **Set Game Language**.
5. Enter `fr` in **Language Code**.
6. Set **Save Player Preference** to **true** for a lasting player choice. Use **false** for a temporary test or a scene-specific change.
7. If needed, add the GUI-image refresh from chapter 8 after the language action.
8. Save the GUI and project, then test the button in the running game.

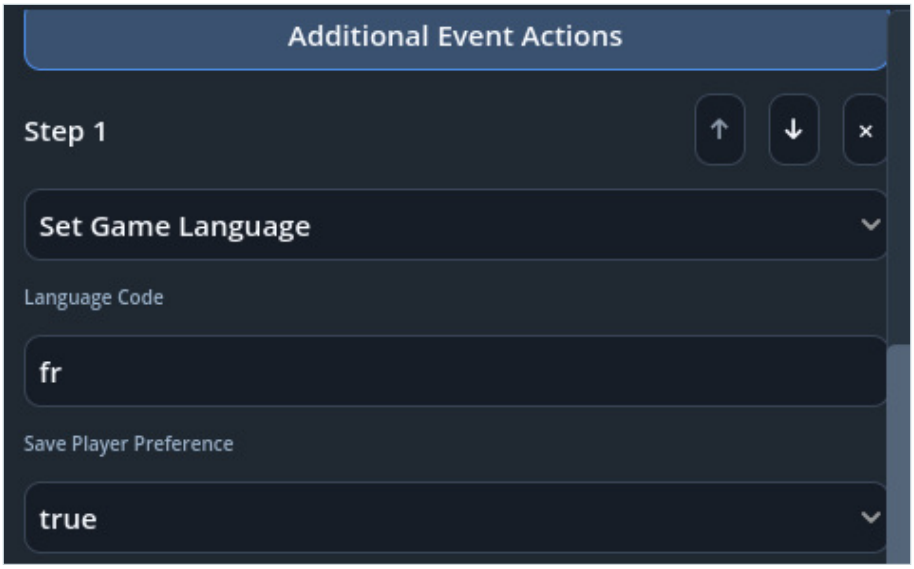



**Figure 25.** Step 1 of the button's click actions selects fr and saves the player preference.

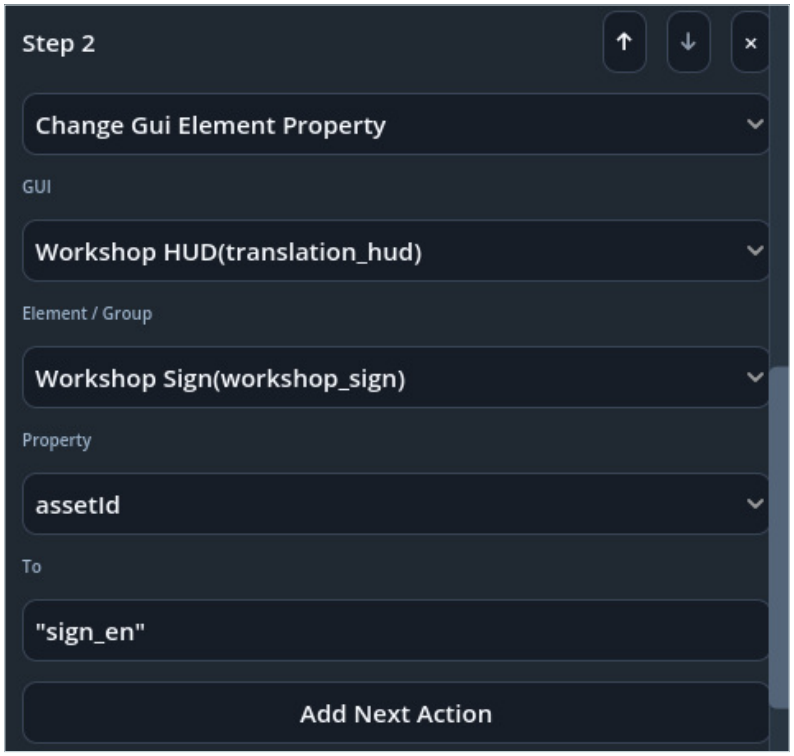



**Figure 26.** Step 2 reapplies the sign's source asset ID. The To field shows a string value, "sign_en".

The **To** field accepts a value, so use the quoted string `"sign_en"` as shown. Replace the GUI, element and asset IDs with your own actual records.

A GUI Designer preview is for checking appearance and supported preview behaviour; it does not need to run game events. Test the actual language-setting event with Play / Restart.

## The related actions

| Action | Main field | Use |
| --- | --- | --- |
| Set Game Language | Language Code; Save Player Preference | Text and translated-asset language. |
| ChangeVoiceLanguage | Language | Choose speech independently. `auto` follows the game language. |
| ChangeSubtitleLanguage | Language | Choose language-tagged movie subtitle cues. |
| ChangeGUIElementProperty | GUI; Element; Property; To | Reapply or change an element property, including the tested image refresh. |

The action selector may display a friendly label or the shared action name. Use the named fields shown for that action.

Only select languages you registered. An unsupported Game Language code is ignored rather than inventing a new language at runtime.

Changing a running game's language does not edit your authored project file. Saving a player preference also does not change the default you ship for new players.

# 17. Change language from scripts and visual scripting

The same language actions are available through the shared action system. Use the normal action node or action-list entry in visual scripting, then set its language fields just as in the GUI event example.

For scripts, keep the entry function supplied by the editor. The following snippets show the relevant calls; you do not need all four language versions in one project.

## GDScript

```
await SetGameLanguage({
    "language": "fr", "persist": true
})
await ChangeVoiceLanguage({"language": "en"})
await ChangeSubtitleLanguage({"language": "fr"})
```

This requests French game text, English speech and French movie subtitle cues. Set `persist` to `false` when trying a temporary game-language change.

To refresh the example GUI sign after selecting the language:

```
await ChangeGUIElementProperty({
    "guiId": "translation_hud",
    "elementId": "workshop_sign",
    "property": "assetId",
    "value": "sign_en"
})
```

The source asset ID stays the same. Its language mapping decides which file is loaded when the element rebuilds.

## PCA Script

```
function run()
    SetGameLanguage({"language": "fr", "persist": true})
    ChangeVoiceLanguage({"language": "auto"})
end
```

### BASIC

```
FUNCTION Run()
    AWAIT SetGameLanguage({"language": "fr", "persist": true})
    AWAIT ChangeVoiceLanguage({"language": "auto"})
END FUNCTION
```

### Lua

```
local Script = {}
function Script.run()
    await(SetGameLanguage({language = "fr", persist = true}))
    await(ChangeVoiceLanguage({language = "auto"}))
end
return Script
```

These use the shared action names and property names. Their punctuation and waiting syntax depend on the scripting language.

The dedicated voice/subtitle actions update the running options, but do not by themselves promise a saved preference. The built-in Options menu's Apply Options saves the combined choices. If you use custom controls, arrange their persistence deliberately; do not assume a previous SetGameLanguage call saved voice changes made afterward.

For spoken-message actions, enter the original source sentence, then the appropriate Voice Key or explicit audio route. Do not switch the source sentence to French inside every branch of the script; let the translation table handle the language choice.

# 18. Regional languages, fonts and layout

## Use a base translation where it helps

A regional code can fall back to its base. For example, a lookup for `fr-ca` can use `fr` text if there is no Canadian French override for that source.

1. Add `fr` and translate a sentence.
2. Add `fr-ca` with a clear name such as **French (Canada)**.
3. Leave the Canadian French box empty for a line that can share the base translation.
4. Add a regional translation only where the wording needs to differ.
5. Register `fr-ca` before trying to select it through Set Game Language.




**Figure 27.** French (Canada) is another registered language. Its empty text field can use the French base translation for this line.

A base-language fallback is not the same as falling back to the project's default language. With no usable translation for an ordinary text lookup, the original source is displayed.

## Give translated text enough room

A short English button can become a longer phrase in another language. Check the actual font and actual GUI rather than comparing character counts in a spreadsheet.

Widen a button, allow suitable wrapping, adjust padding or choose shorter wording that still means the same thing. Avoid shrinking every translated caption until the interface is hard to read.

Check choice lists, inventory names, save slots, journal headings, tooltip edges and narrow mobile layouts. Check at the largest player UI Font Scale you intend to support. A line that fits at the designer's normal size can clip after the player enlarges text.

## Choose fonts that contain the characters

A font needs the glyphs used by the language: accents, punctuation and any non-Latin writing system. Test with real sentences, not only the language's name in a selector.

If boxes or missing symbols appear, check the assigned font and any suitable fallback-font support before rewriting correct text. A translated font asset can be mapped where the loading path supports it, but verify the actual GUI or text renderer that uses the font.

For right-to-left scripts, test whole paragraphs, mixed numbers, punctuation, alignment and keyboard entry in your actual GUI. Adding a language code alone does not prove that every custom field has the right layout behaviour.

## Keep images and text together in testing

A translated sign can have different word lengths even when its image dimensions match. Check its fit mode, crop, transparent border and legibility in the game. An asset mapping can choose the correct file and still look wrong because the image is stretched or cut off.

# 19. Test the translated game before release

A successful CSV import proves that the accepted rows passed the import checks. It does not prove the translation is complete, natural or playable.

## Do one full language pass

1. Save Translations and Save Project.
2. Set the intended test language and start a fresh preview.
3. Check the title, startup presentation, menu and Options.
4. Play the opening room and its interaction labels.
5. Read every dialogue choice, including alternate branches and repeated visits.
6. Test inventory, journal, quest updates and save/load displays.
7. Check messages produced by scripts and visual actions.
8. Check placeholders with different player names and variable values.
9. Check translated images, recordings and movies at the places they are loaded.
10. Switch language during play and look for controls that keep an earlier image or caption.
11. Check the intended voice/subtitle combinations and lip sync.
12. Check text size and layout on the smallest target display you support.

Use the screenshots as a guide to the expected result, not as a substitute for playing your own game. Your custom GUI may route text differently from the built-in controls.

## Then check the distributable build

1. Run the normal project validation and resolve missing resource references.
2. Save the project and build the intended target.
3. Keep the build's supporting files with the executable/application.
4. Launch it as a fresh player to check the authored default language.
5. Change language and apply options, close the game, then reopen it to check the saved choice.
6. Load a save made earlier and check the text/resources the game reconstructs.
7. Test each shipped language, including a missing translation and an expected fallback.

Do not remove translated images or recordings simply because they are not directly assigned to a visible object. A translation mapping can be their reference. Check any resource-cleanup proposal against the mapping tables before accepting it.

If your game stores a piece of already-expanded text in a save, switching language may not recreate that text automatically. Test those saved messages or custom logs explicitly. Store stable game state and regenerate display text where your design allows it.

### Keep a short issue list

For each issue, record the language code, scene/screen, source line or asset ID, what appeared, and what should appear. Add a screenshot when a layout problem is hard to describe. This gives the author or translator a specific correction to make.

# 20. Quick fixes and a final checklist

### When something looks wrong

| Symptom | First checks |
|---|---|
| The editor changed language but the game did not | Use Game Translations and Game Language, not the editor-language toolbar control. |
| A translated line still appears in English | Exact source spelling; saved translation; active Game Language; supported display path. |
| One version of a sentence translates and another does not | Compare punctuation, spaces and capital letters. They may be different keys. |
| A French CSV will not import | Select French in CSV Language; keep every row's code and all required columns intact. |
| Apply Changes is disabled | Read the preview. All rows may be unchanged, blank, conflicting or invalid. |
| A blank CSV cell did not erase an old translation | Expected. Clear the translation in the editor to remove it. |
| New project wording seems to disappear on import | Check conflicts and the original baseline; do not enable overwrite without review. |
| Player names or scores stopped updating | Preserve @PlayerName, $$variable_name and other supported markers exactly. |
| The sign stays in the old language | Check its mapping and the loaded image control. Use the tested reapply-asset step for GUI images. |
| French text plays English speech | Check Voice Language, missing-language fallback, explicit audio assignments and Voice Key routing. |
| The correct voice plays but the mouth is wrong | Use timings for that exact translated recording and check portrait/character mappings. |

| Symptom | First checks |
| --- | --- |
| Only movie captions changed | ChangeSubtitleLanguage affects tagged movie cues; ordinary text needs Set Game Language. |
| A new default is ignored in an installed build | Check an existing saved player preference. Compare a fresh player with the embedded preview. |
| Accents become strange characters after a return | Re-export the spreadsheet as proper UTF-8 CSV/TSV. |
| Text turns into boxes or clips at the edges | Font glyph coverage, available width, wrapping, margins and player font scale. |

## Before sending a translation out

- Save a known project revision and an untouched export.
- Confirm the target language code.
- Give the translator context, a glossary and placeholder instructions.
- Keep columns and baseline values intact.
- Assign shared source rows only once.

## Before accepting a return

- Save a project checkpoint.
- Select the matching CSV Language.
- Review incoming wording, conflicts and invalid rows.
- Apply only the work you mean to accept.
- Save Translations, then Save Project.

## Before calling a language finished

- Play the actual screens, conversations and puzzle branches.
- Check menu text, messages, choices, item names and journal entries.
- Check voices, lip sync, images and movie subtitles separately.
- Check missing entries and regional fallbacks on purpose.
- Test a fresh player and a returning player in the distributable build.
- Have a fluent reader review the game in context.

Keep the source text steady once a translation round starts. Keep the returned files. Test one complete language at a time. Those three habits prevent most of the avoidable cleanup.