Localizer Easy – Complete Documentation

A powerful, lightweight, high-performance CSV-based localization system for Unity that allows you to easily manage multi-language translations without writing code.

Table of Contents

Core Features

Localization System:

- **No coding required** - Drag and drop setup in Unity Inspector
- CSV-based localization with UTF-8 support and automatic file detection
- ScriptableObject settings (drag folder to auto-detect CSV files)
- Multi-language support via SystemLanguage enum with Editor/Runtime separation
- Cross-platform compatibility (all Unity-supported platforms via TextAsset)
- Smart caching system (10,000 entry limit with FIFO strategy) for optimized performance

Editor Integration:

- TextMeshPro integration with automatic text updates

- Editor preview language support with live preview
- Searchable key selection window with fuzzy search and keyboard navigation
- CSV validation tool with detailed error reporting
- Translation editing in Inspector with save/cancel functionality
- Manual CSV reload buttons in Inspector and Project Settings

Runtime Features:

- Smart caching system (10,000 entry limit with FIFO strategy)
- Automatic component updates (including inactive objects) when language changes
- Language change event system (OnLanguageChanged)
- Parameter substitution with Get() method
- Dynamic CSV file name and key assignment via code
- Dynamic CSV loading at runtime (LoadCSVFromString) - load translations from server, mods, or external files
- Complete CSV replacement support for hot-fixing and content updates
- Consistent API behavior across all methods

Performance & Code Quality:

- Zero-allocation lookups with ValueTuple cache keys
- Optimized dictionary lookups and memory-efficient CSV parsing
- Clean architecture (Manager, Loader, Component separation)
- Comprehensive XML documentation and error handling
- Cross-version compatibility (string-based language storage)

Quick Start

Try the Demo: Open `Assets/LocalizerEasy/Demo/LocalizerEasyDemo.unity` to see a working example with language switching and dynamic text updates.

Get started with LocalizerEasy in just 3 simple steps:

1. **Create CSV Files** - Create CSV files in your desired folder (you can copy and modify CSV files from the Demo folder)
2. **Configure Settings** - Drag the folder containing CSV files to Project Settings
3. **Add Component** - Attach Localizer component to TextMeshPro objects

Step 1: Create CSV Files

Create UTF-8 encoded CSV files in your desired folder location. You can copy and modify the CSV files from `Assets/LocalizerEasy/Demo/CSV/` as a starting template.

Note: CSV files do NOT need to be placed in Resources or StreamingAssets folders. They can be anywhere in your Assets folder.

CSV Format Example:

```
Key,English,German,Spanish
start_game,Start Game,Spiel starten,Iniciar juego
pause,Pause,Pause,Pausa
options,Options,Optionen,Opciones
welcome_msg,Welcome to this world!,Willkommen in dieser Welt!,
¡Bienvenido a este mundo!
coins_earned,"You earned {0} coins, {1} EXP!","Du hast {0} Münzen und
{1} EXP erhalten!","¡Ganaste {0} monedas y {1} EXP!"
multiline_text,"Line 1<br>Line 2<br>Line 3","Zeile 1<br>Zeile 2<br>Zeile
3","Línea 1<br>Línea 2<br>Línea 3"
quote_example,"He said ""Hello, friend!""","Er sagte ""Hallo,
Freund!""","Él dijo ""Hola, amigo!"""
toggle_option,On/Off,Ein/Aus,Activado/Desactivado
file_path,C:\Users\Game\Save.dat,C:\Benutzer\Spiel\Speichern.dat,C:\Usua
rios\Juego\Guardar.dat
```

Special Characters Handling:

Case	Example	Note
Contains comma	`"Hello, World"`	Wrap entire text in double quotes
Contains quotes	`"He said ""Hello"""`	Use two consecutive double quotes `""` (not `\"`)
Contains both	`"Hello, he said ""Hi"""`	Combine both rules
Contains slash	`Yes/No or On/Off`	Can be used directly, no special handling needed
Contains backslash	`C:\Path\File.txt`	Can be used directly, no special handling needed

CSV Format Rules:

- First column must be `Key`
- Subsequent columns should match `SystemLanguage` enum names exactly
- **System Language Reference:** See Unity SystemLanguage Documentation
- Language column order is customizable (e.g., `Key,English,German,Spanish` or `Key,German,Spanish,English`)
- Use `<br>` or `<BR>` for line breaks (case-insensitive)
- Always use UTF-8 encoding

Step 2: Configure Settings

Go to **Edit > Project Settings > Localizer Easy** (or **Tools > Localizer Easy > Open Settings**)

CSV Folder: Drag a folder containing CSV files to the **CSV Folder** field

- All `.csv` files in this folder will be automatically detected and added to the file list
- CSV files can be anywhere in your Assets folder

- Files are referenced as TextAssets, so they'll be included in builds automatically
- CSV content is automatically loaded when entering Play Mode

Click "Reload All CSV" to load immediately.

Show Logs (Optional): Toggle to show/hide CSV loading messages in Console

Reload All CSV (Optional): Click to rescan folder and reload all CSV files after editing CSV

Settings Features:

- **Automatic CSV Detection**: Just drag a folder, all CSV files are detected and added to the list
- **Automatic Loading**: CSV content is loaded when entering Play Mode (or click "Reload All CSV" to load immediately)
- **Build Inclusion**: CSV files are automatically included in builds
- **CSV Validation**: Use "Check CSV Files" button to validate all CSV files

Settings Location:
The settings file is automatically created at `Assets/Resources/LocalizerEasySettings.asset`

Important: The settings file MUST be in a Resources folder to be loaded at runtime. If you have special requirements, you can manually drag `LocalizerEasySettings.asset` to other Resources folders.

Step 3: Add Component to UI

1. Select a GameObject with a **TextMeshPro** component
2. Click **Add Component** → Search for **Localizer**
3. In Inspector:
    - **Preview Language**: Use the dropdown to instantly preview all scene text components in different languages (Edit Mode only)
    - **CSV File Name**: Select your CSV file (e.g., "ui")

- **Key**: Select the translation key (click search button for search)
- **Translation Preview (Editable)**: View all language translations in this section. Edit any text and click Save to write changes directly to the CSV file with automatic reload

4. Press Play - text will automatically display in the current language!

Step 4: Switch Languages in C# (Optional)

```
using LocalizerEasy;

// Switch language
LocalizerManager.CurrentLanguage = SystemLanguage.English;
```

Installation & Setup

LocalizerEasy is a Unity package that can be imported directly into your project.

Storage Structure

```
Dictionary<string, Dictionary<string, Dictionary<SystemLanguage,
string>>>
// CSV Name -> Key -> Language -> Translation
```

Runtime Cache

- **Purpose**: Fast lookup for frequently accessed translations
- **Mechanism**: Caches `(csv, key, language)` combinations using ValueTuple for efficient key generation
- **Size Limit**: Maximum 10,000 entries (FIFO strategy - removes oldest entries first when exceeded)
- **Auto-Clear**: Cleared when language changes
- **Optimization**: Uses ValueTuple to avoid string concatenation overhead, improving performance and reducing GC pressure

API Reference

Get Translation

```
// Basic translation (uses current language)
string text = LocalizerManager.Get("ui", "start_game");
textMeshPro.text = text;

// With parameters (uses current language)
string text = LocalizerManager.Get("ui", "coins_earned", 100, 50);
textMeshPro.text = text;
// CSV: coins_earned,"You earned {0} coins, {1} EXP!"
// Output: "You earned 100 coins, 50 EXP!"
```

Note:

- `LocalizerManager.Get()` uses the current language (`LocalizerManager.CurrentLanguage`)
- To change language, set `LocalizerManager.CurrentLanguage = SystemLanguage.English;`
- Returns the key itself if translation is not found
- For dynamic component configuration, see Dynamic Component Configuration section

Language Management

```
// Get current language
SystemLanguage current = LocalizerManager.CurrentLanguage;

// Set language (automatically saves to PlayerPrefs and updates all
components)
LocalizerManager.CurrentLanguage = SystemLanguage.English;

// Language change event (optional - components are automatically
updated)
LocalizerManager.OnLanguageChanged += () => {
    Debug.Log($"Language changed to:
{LocalizerManager.CurrentLanguage}");
};
```

System Language Reference: See Unity SystemLanguage Documentation

CSV Management

```
// Get available CSV names
string[] csvFileNames = LocalizerManager.GetAvailableCSVs();

// Get keys for a CSV
string[] keys = LocalizerManager.GetKeysForCSV("ui");

// Dynamically load CSV from string (runtime only)
string csvContent = "Key,English,German\nstart_game,Start Game,Spiel
starten";
bool success = LocalizerManager.LoadCSVFromString("custom", csvContent,
updateComponents: true);

// Replace existing CSV at runtime (e.g., update from server)
string updatedContent = File.ReadAllText(csvPath,
System.Text.Encoding.UTF8);
LocalizerManager.LoadCSVFromString("ui", updatedContent); // Replaces
built-in "ui" CSV
```

Dynamic Component Configuration

You can programmatically set CSV file name and key for `Localizer` components, allowing text to automatically update when language changes:

```
using LocalizerEasy;
using TMPro;

// Get Localizer component from TMP_Text GameObject
var localizerComponent = rewardText.GetComponent<Localizer>();
if (localizerComponent == null)
{
    // Add component if it doesn't exist
    localizerComponent = rewardText.gameObject.AddComponent<Localizer>
();
}

// Set CSV file name and key (simple translation)
localizerComponent.Get("ui", "start_game");

// Set CSV file name, key, and format parameters (with parameters)
int coins = 100;
int exp = 50;
localizerComponent.Get("ui", "coins_earned", coins, exp);

// Text will automatically update when language changes!
```

Benefits:

- Text automatically updates when `LocalizerManager.CurrentLanguage` changes
- No need to manually set CSV file name and key in Inspector
- Supports format parameters for dynamic content
- Works with both simple translations and formatted translations

Dynamic CSV Loading at Runtime

You can dynamically load CSV content from strings at runtime, enabling powerful features like downloading translations from a server, loading user mods, or updating content without rebuilding the game.

Method:

```
bool LocalizerManager.LoadCSVFromString(string csvFileName, string csvContent, bool updateComponents = true)
```

Parameters:

- `csvFileName` : Name identifier for this CSV (used in `Get()` calls). If this name already exists, it will be **completely replaced**.
- `csvContent` : CSV file content as string (must be UTF-8 encoded, follows the same format rules as regular CSV files)
- `updateComponents` : Whether to automatically update all Localizer components after loading (default: true)

Returns: `true` if loading succeeded, `false` otherwise

Use Cases:

```
using System.IO;
using UnityEngine;
using LocalizerEasy;

// Example 1: Load translations from external file
string csvPath = Path.Combine(Application.persistentDataPath, "custom_translations.csv");
if (File.Exists(csvPath))
{
    string csvContent = File.ReadAllText(csvPath, System.Text.Encoding.UTF8);
```

```
    bool success = LocalizerManager.LoadCSVFromString("custom",
csvContent);

    if (success)
    {
        // Now you can use translations from this CSV
        string text = LocalizerManager.Get("custom", "some_key");
    }
}

// Example 2: Download translations from server and replace built-in CSV
IEnumerator DownloadTranslations()
{
    UnityWebRequest www =
UnityWebRequest.Get("https://yourserver.com/translations/ui.csv");
    yield return www.SendWebRequest();

    if (www.result == UnityWebRequest.Result.Success)
    {
        string serverCsvContent = www.downloadHandler.text;
        // This will completely replace the built-in "ui" CSV
        bool success = LocalizerManager.LoadCSVFromString("ui",
serverCsvContent);

        if (success)
        {
            Debug.Log("Translations updated from server!");
            // All UI components using "ui" CSV will be automatically
updated
        }
    }
}

// Example 3: Load user mod translations
string modPath = Path.Combine(Application.persistentDataPath,
"mods/my_mod.csv");
```

```
if (File.Exists(modPath))
{
    string modContent = File.ReadAllText(modPath,
System.Text.Encoding.UTF8);
    LocalizerManager.LoadCSVFromString("mod_translations", modContent);

    // Use mod translations
    myText.text = LocalizerManager.Get("mod_translations", "mod_key");
}

// Example 4: Create CSV content programmatically
string dynamicCsv = "Key,English,German,Spanish\n" +
                    "dynamic_key_1,Hello,Hallo,Hola\n" +
                    "dynamic_key_2,Goodbye,Auf Wiedersehen,Adiós";
LocalizerManager.LoadCSVFromString("dynamic", dynamicCsv);
```

Important Notes:

- If `csvFileName` already exists (e.g., "ui"), all existing translations will be **completely replaced** with the new content
- CSV content must follow the same format rules as regular CSV files (first column: "Key", subsequent columns: SystemLanguage enum names)
- The loaded CSV behaves exactly like built-in CSV files - you can use `Get()` to retrieve translations and components will auto-update when language changes
- Cache is automatically cleared for the affected CSV to ensure new translations are used immediately
- If `updateComponents` is true, all Localizer components in the scene will be updated after loading

Performance Best Practices - Batch Loading:

When loading multiple CSV files, it's important to optimize performance by avoiding unnecessary component updates. Use `updateComponents: false` for all but the last file:

```
// BAD: Updates components 5 times (poor performance)
LocalizerManager.LoadCSVFromString("ui", uiContent, true);         //
Updates all components
LocalizerManager.LoadCSVFromString("story", storyContent, true);   //
Updates all components
LocalizerManager.LoadCSVFromString("items", itemsContent, true);   //
Updates all components
LocalizerManager.LoadCSVFromString("quests", questContent, true);  //
Updates all components
LocalizerManager.LoadCSVFromString("dialogs", dialogContent, true);//
Updates all components
// Result: UpdateAllComponents() called 5 times - very inefficient!

// GOOD: Updates components only once (optimal performance)
LocalizerManager.LoadCSVFromString("ui", uiContent, false);
LocalizerManager.LoadCSVFromString("story", storyContent, false);
LocalizerManager.LoadCSVFromString("items", itemsContent, false);
LocalizerManager.LoadCSVFromString("quests", questContent, false);
LocalizerManager.LoadCSVFromString("dialogs", dialogContent, true); //
Only update on last file
// Result: UpdateAllComponents() called only once - efficient!

// ALTERNATIVE: Manual control for even more flexibility
LocalizerManager.LoadCSVFromString("ui", uiContent, false);
LocalizerManager.LoadCSVFromString("story", storyContent, false);
LocalizerManager.LoadCSVFromString("items", itemsContent, false);
LocalizerManager.LoadCSVFromString("quests", questContent, false);
LocalizerManager.LoadCSVFromString("dialogs", dialogContent, false);
// Perform other operations if needed...
LocalizerManager.UpdateAllComponents(); // Update once when ready
```

Real-World Example - Download Multiple Files from Server:

```
IEnumerator DownloadAllTranslations()
{
```

```
    string[] csvNames = { "ui", "story", "items", "quests", "dialogs" };
    string baseUrl = "https://yourserver.com/translations/";

    for (int i = 0; i < csvNames.Length; i++)
    {
        string csvName = csvNames[i];
        UnityWebRequest www = UnityWebRequest.Get(baseUrl + csvName +
".csv");
        yield return www.SendWebRequest();

        if (www.result == UnityWebRequest.Result.Success)
        {
            string csvContent = www.downloadHandler.text;

            // Only update components on the last file
            bool isLastFile = (i == csvNames.Length - 1);
            bool success = LocalizerManager.LoadCSVFromString(csvName,
csvContent, isLastFile);

            if (success)
            {
                Debug.Log($"Downloaded: {csvName}.csv");
            }
        }
        else
        {
            Debug.LogError($"Failed to download {csvName}.csv:
{www.error}");
        }
    }

    Debug.Log("All translations downloaded and loaded!");
}
```

Performance Impact:

- **Without optimization**: Loading 5 CSV files = 5x `UpdateAllComponents()` calls
- **With optimization**: Loading 5 CSV files = 1x `UpdateAllComponents()` call
- **Result**: Up to 5x faster loading when dealing with multiple CSV files
- **Recommended**: Always use `updateComponents: false` when batch loading, and update once at the end

Editor Features

Inspector Integration

- **CSV File Name Dropdown**: Select from available CSV files
- **Key Dropdown**: Select translation key with search button
  - Click search button to open searchable popup window
  - Fuzzy search support (searches both key names and translations)
- **Preview Language**: Test translations in different languages without playing
- **Translation Preview**: View all language translations in Inspector and edit them directly with save functionality
- **Reload All CSV Button**: Rescan folder and reload all CSV files
- **Settings Button**: Quick access to Project Settings

CSV Validation

Use **Check CSV Files** button in Project Settings to validate:

- Column format (first column must be "Key")
- Language names (must match SystemLanguage enum)
- Duplicate keys
- Missing translations
- CSV format errors (quotes, commas, etc.)

Translation Editing

In Inspector preview section:

- Edit translations directly in Inspector
- Save changes back to CSV file
- Cancel changes if needed
- Changes are immediately reflected

Platform Support

Supported Platforms

- **Windows / Mac / Linux** (Editor & Standalone)
- **Android** (Full support via TextAsset)
- **iOS** (Full support via TextAsset)
- **WebGL** (Full support via TextAsset)
- **All Unity-supported platforms** (CSV files are included in build automatically)

Language Persistence

Editor Mode:

- Uses `EditorPrefs` to save preview language
- Key: `LocalizerEasy_PreviewLanguage`
- Does NOT use PlayerPrefs (to avoid affecting actual game settings)

Runtime Mode:

- Uses `PlayerPrefs` to save language preference
- Key: `LocalizerEasy_CurrentLanguage`
- Format: String (cross-version compatible)
- Falls back to `Application.systemLanguage` if no saved preference

Troubleshooting

Text Shows Key Instead of Translation

Possible Causes:

1. CSV file doesn't exist or not loaded
2. CSV doesn't have column for current language
3. Key doesn't exist in CSV
4. CSV format is incorrect

Solutions:

- Check Console for error messages
- Verify CSV file is in the folder specified in Settings
- Click "Reload All CSV" button in Inspector or Project Settings
- Use "Check CSV Files" in Project Settings to validate CSV format

Inspector Doesn't Show CSV Dropdown

Solutions:

- Make sure CSV files are added in Project Settings
- Click "Reload All CSV" button in Inspector or Project Settings
- Check Console for loading errors
- Verify CSV folder is correctly set in Settings

CSV Not Loading

Solutions:

- Ensure CSV is UTF-8 encoded
- Check CSV format matches specification (first column must be "Key")

- Verify CSV files are in the specified folder
- Check Console for detailed error messages

Language Not Persisting

Editor Mode:

- Preview language is saved to EditorPrefs automatically
- Check if EditorPrefs is working

Runtime Mode:

- Language preference is saved to PlayerPrefs automatically
- Check PlayerPrefs settings in build configuration
- Verify platform supports PlayerPrefs

Performance Issues

If you have thousands of keys:

- System uses smart caching (only caches accessed translations)
- Cache limit: 10,000 entries (FIFO strategy - removes oldest entries when exceeded)
- Cache uses ValueTuple keys for optimal performance

Components Not Updating

If components don't update when language changes:

- All components (including inactive ones) are automatically updated when language changes
- Components automatically update when they become active (via `OnEnable()`)
- If a component still shows old translation, check Console for error messages
- Verify CSV file and key are correctly set in Inspector

FAQ

Q: How do I add more languages?

A: Add new columns to your CSV files with SystemLanguage enum names:

```
Key,English,Korean,French,German
start_game,Start Game,게임 시작,Démarrer le jeu,Spiel starten
```

Q: Can CSV files be in subfolders?

A: Yes! When you drag a folder to Settings, all CSV files in that folder and subfolders are automatically detected.

Q: Will CSV files be included in builds?

A: Yes! CSV files are referenced as TextAssets in the ScriptableObject, so they're automatically included in builds.

Q: How do I change CSV location?

A: Simply drag a different folder to the CSV Folder field in Project Settings. All CSV files will be updated automatically.

Q: Can I use CSV files from anywhere in Assets?

A: Yes! CSV files can be anywhere in your Assets folder. Simply drag a folder containing CSV files to the **CSV Folder** field in Project Settings. All CSV files in that folder (and subfolders) will be automatically detected and added to the system.

Note: CSV files do NOT need to be placed in Resources or StreamingAssets folders. They are automatically included in builds via ScriptableObject references.

Q: What happens if CSV is missing a translation?

A: The system displays the Key itself. In Editor mode, you'll see a warning in Console. You can edit translations directly in Inspector.

Q: How do I refresh CSV files after editing?

A:

- Click "Reload All CSV" button in Inspector or Project Settings
- Both buttons will rescan the CSV folder and reload all CSV files

Q: How do I update all components after CSV reload?

A:

- **Automatic**: When clicking "Reload All CSV" button, all scene components are automatically updated after reloading CSV files
- **Language change**: All components (including inactive ones) are automatically updated when language changes via `CurrentLanguage` setter
- **Component enabled**: Each component automatically updates when it becomes active (via `OnEnable()`)

Q: Does it work with Unity's Localization Package?

A: No, LocalizerEasy is a standalone solution. You can use either LocalizerEasy or Unity's Localization Package, but not both together.

Q: Can I use this in a team project?

A: Yes! CSV files are version-controlled and can be easily shared. Settings are stored in ScriptableObject, which is also version-controlled.

Q: Can I set CSV file name and key programmatically?

A: Yes! Use `Get(csvFileName, key, params...)` method on the `Localizer` component. The text will automatically update when language changes, even if CSV file name and key are not set in Inspector.

```
var localizerComponent = textComponent.GetComponent<Localizer>();
localizerComponent.Get("ui", "coins_earned", 100, 50);
// Text will automatically update when language changes!
```

Q: Can I dynamically load CSV files at runtime?

A: Yes! Use `LoadCSVFromString()` to load CSV content from strings at runtime:

```
// Load from file
string csvContent = File.ReadAllText(csvPath,
System.Text.Encoding.UTF8);
bool success = LocalizerManager.LoadCSVFromString("custom", csvContent);

// Replace existing CSV (e.g., update "ui" from server)
LocalizerManager.LoadCSVFromString("ui", newContent); // Completely
replaces existing "ui" CSV
```

This is useful for:

- Downloading updated translations from a server
- Loading user-created language packs or mods
- Hot-fixing translation errors without rebuilding the game
- A/B testing different translations

Q: Can I update translations without restarting the game?

A: Yes! Use `LoadCSVFromString()` to replace any CSV at runtime. All Localizer components will be automatically updated:

```
// Download and apply new translations
string newTranslations = await DownloadFromServer();
LocalizerManager.LoadCSVFromString("ui", newTranslations);
// All UI text is immediately updated!
```

Q: How do I load multiple CSV files efficiently?

A: Use `updateComponents: false` for all files except the last one to avoid redundant updates:

```
// Load multiple CSV files (only update UI once at the end)
LocalizerManager.LoadCSVFromString("ui", uiContent, false);
LocalizerManager.LoadCSVFromString("story", storyContent, false);
LocalizerManager.LoadCSVFromString("items", itemsContent, false);
LocalizerManager.LoadCSVFromString("dialogs", dialogContent, true); //
Update on last file

// This is 4-5x faster than updating after each file!
```

Why? Each `UpdateAllComponents()` call searches through all GameObjects in the scene. When loading 5 CSV files with `updateComponents: true`, you're doing this expensive operation 5 times. By using `false` and updating only once at the end, you get the same result with much better performance.

Rule of thumb: Always use `updateComponents: false` when batch loading, and update once at the end.

Code Examples

Demo Scene: `Assets/LocalizerEasy/Demo/LocalizerEasyDemo.unity`
Demo Script: `Assets/LocalizerEasy/Demo/LocalizerEasyDemo.cs`

Open the demo scene to see a complete working example with language switching buttons and dynamic text updates.

Complete Usage Example

```
using UnityEngine;
using UnityEngine.UI;
using LocalizerEasy;
using TMPro;

namespace LocalizerEasy.Demo
{
    public class LocalizerEasyDemo : MonoBehaviour
    {
        public Button BtnEnglish;
        public Button BtnGerman;
        public Button BtnSpanish;

        public TMP_Text WelcomeText;
        public TMP_Text RewardText;

        // Localizer component reference (optional - can be set in
Inspector or GetComponent)
        private Localizer _rewardTextLocalizer;

        void Start()
        {

            // Set initial language
            LocalizerManager.CurrentLanguage = SystemLanguage.English;

            BtnEnglish.onClick.AddListener(() =>
            {
                LocalizerManager.CurrentLanguage =
SystemLanguage.English;
            });
            BtnGerman.onClick.AddListener(() =>
            {
```

```
                LocalizerManager.CurrentLanguage =
SystemLanguage.German;
            });
            BtnSpanish.onClick.AddListener(() =>
            {
                LocalizerManager.CurrentLanguage =
SystemLanguage.Spanish;
            });

            // Get Localizer component from RewardText GameObject
            // RewardText has a Localizer component attached in the
Inspector (empty settings).
            // Settings are assigned via C# code so the text updates
automatically when the language changes.
            _rewardTextLocalizer = RewardText.GetComponent<Localizer>();
            if (_rewardTextLocalizer == null)
            {
                // If component doesn't exist, add it automatically
                _rewardTextLocalizer =
RewardText.gameObject.AddComponent<Localizer>();
            }

            // Set translation dynamically using Get (instance method)
            // This will automatically update when language changes
            // CSV: coins_earned,"You earned {0} coins, {1} EXP!"
            int coins = 100;
            int exp = 50;
            _rewardTextLocalizer.Get("ui", "coins_earned", coins, exp);

            // Subscribe to language change event
            LocalizerManager.OnLanguageChanged += UpdateTexts;

            // Initial text update
            UpdateTexts();
        }
```

```
        /// <summary>
        /// Updates text components when language changes
        /// </summary>
        private void UpdateTexts()
        {
            // Basic translation (uses current language)
            // CSV: welcome_msg,Welcome to this world!
            WelcomeText.text = LocalizerManager.Get("story",
"welcome_msg");

            // RewardText will be automatically updated by Localizer
component
            // But if you need to update format parameters, call Get
again
            // Example: _rewardTextLocalizer.Get("ui", "coins_earned",
coins, exp);
        }

        void OnDestroy()
        {
            // Unsubscribe from language change event to prevent memory
leaks
            LocalizerManager.OnLanguageChanged -= UpdateTexts;
        }
    }
}
```