



Language

Documentation

Creator: Lucas Gomes Cecchini

Online Name: AGAMENOM

Overview

This document will help you to use **Assets Language** for **Unity**.

With it you can translate your game into several languages, automatic selection of languages (if you have a file available) and a simple text file to interpret, making it possible for anyone to translate the game into a new language.

It also has simple features to translate images and sound.

Instructions

You can get more information from the Playlist on YouTube:

https://youtube.com/playlist?list=PL5hnfx09yM4JkAyxrZWaFjhO3NMWxP_1F

Script Explanations

Class

LanguageClassTools

This script provides several utility methods for handling language-related operations within a Unity environment. Let's break down each method and variable:

Variables:

- **folderNameInUnity**: String variable used to specify the folder name in Unity Editor.
- **folderNameInBuild**: String variable used to specify the folder name in a build.
- **jsonNameInUnity**: String variable used to specify the JSON file name in Unity Editor.
- **jsonNameInBuild**: String variable used to specify the JSON file name in a build.
- **path**: Represents the directory path where language files are searched.
- **standardFile**: A reference identifier used to locate a default language file.

Methods:

1. **GetFolderPath(string folderNameInUnity, string folderNameInBuild)**
 - **Usage**: Generates a folder path based on conditions (Unity Editor vs. build).
 - **Return**: Returns the appropriate folder path based on the environment.
2. **GetJsonPath(string jsonNameInUnity, string jsonNameInBuild)**
 - **Usage**: Generates a JSON file path based on conditions (Unity Editor vs. build).
 - **Return**: Returns the appropriate JSON file path based on the environment.
3. **FindDefaultLanguageFile(string path, string standardFile)**
 - **Usage**: Searches for a default language file in a directory.
 - **Return**: Returns the path to the matching file or null if not found.
4. **RemoveContentWithinCurlyBraces(string input)**
 - **Usage**: Removes content within curly braces from a string.
 - **Return**: Returns the modified string with content removed.
5. **ExtractTextWithinBraces(string input)**
 - **Usage**: Extracts text within curly braces from a string.
 - **Return**: Returns the extracted text or an empty string if not found properly.
6. **ExtractIntValue(string line, string identifier)**
 - **Usage**: Extracts an integer value from a string based on an identifier.
 - **Return**: Returns the extracted integer value or 0 if not found or invalid.
7. **ExtractFloatValue(string line, string identifier)**
 - **Usage**: Extracts a float value from a string based on an identifier.
 - **Return**: Returns the extracted float value or 0.0f if not found or invalid.
8. **ExtractBooleanValue(string line, string identifier, bool defaultValue)**
 - **Usage**: Extracts a boolean value from a string based on an identifier.
 - **Return**: Returns the extracted boolean value or the provided default value.
9. **ExtractLanguageName(string filePath)**
 - **Usage**: Retrieves the language name from a language file.
 - **Return**: Returns the extracted language name or sets a default if not found.

How to use:

- **Importing the class**: Import this class into other scripts by using `using` statement at the top of the script.

- **Calling methods:** Call these methods by referencing the class name (`LanguageClassTools.MethodName`).

For instance:

```
```\nstring folderPath = LanguageClassTools.GetFolderPath("FolderNameInUnity/", "FolderNameInBuild/");\nstring jsonPath = LanguageClassTools.GetJsonPath("JsonNameInUnity.json", "JsonNameInBuild.json");\n```
```

These methods can assist in managing language-related functionalities within a Unity project, such as locating language files, extracting data, and handling language-specific operations.

## LanguageClassToolsEditor

This script provides several utility methods specifically designed for the Unity Editor environment. Let's dissect each variable and method:

### Variables:

- **groupsID, textID, commentsID**: Lists used to store different types of language-related data entries.
- **componentType**: Represents different types of components.
- **yourImage**: A Texture2D variable used to store images loaded from file paths.
- **pathToImage**: String variable representing the path to images associated with component types.

### Methods:

1. **DrawColoredButton(string label, float textSize, float buttonSize, Color buttonColor, Color textColor, System.Action clickAction)**
  - **Usage**: Draws a colored button in the Unity Editor GUI.
  - **Parameters**: Label text, text size, button size, button color, text color, action on button click.
2. **DrawBox(System.Action content, Color Color)**
  - **Usage**: Draws a colored box in the Unity Editor GUI.
  - **Parameters**: Content to be placed inside the box, color of the box.
3. **GetDuplicateIDs(List<LanguageSaveGroups> groupsID, List<LanguageSaveID> textID, List<LanguageSaveComments> commentsID)**
  - **Usage**: Finds and returns duplicate IDs in different data lists.
  - **Return**: Dictionary containing duplicate IDs and their occurrences.
4. **DefaultSettings(ref List<LanguageSaveGroups> groupsID, ref List<LanguageSaveID> textID, ref List<LanguageSaveComments> commentsID)**
  - **Usage**: Sets default values for language-related data lists.
5. **ExtractBooleanValue(string lineWithoutCurlyBraces, string keyword)**
  - **Usage**: Extracts a boolean value from a string based on a specified keyword.
  - **Return**: Boolean value extracted from the string.
6. **ViewComponent(float componentType)**
  - **Usage**: Displays an image representing the component type in the Unity Editor.
  - **Displays**: Images associated with different component types.
7. **ClearExistingLanguageSaveIDValues(LanguageSaveID existingLanguageSaveID)**
  - **Usage**: Resets or clears the properties of an existing LanguageSaveID entry.
8. **IsIDInTextIDList(float ID, ref List<float> IDs)**
  - **Usage**: Checks if a given ID exists within a list of IDs (TextID).
9. **LoadDataFromFile(ref List<float> IDs)**
  - **Usage**: Loads data from a JSON file into the provided list of IDs.
10. **SaveAdjustRect(ref List<AdjustRectList> adjustRectList)**
  - **Usage**: Saves AdjustRectList data by extracting relevant properties.
11. **ExportIDsToAdjustRect(List<AdjustRectList> adjustRectList)**
  - **Usage**: Associates IDs with corresponding entries in the AdjustRectList.

### How to use:

- **Importing the class**: Use ``#if UNITY_EDITOR`` to ensure these methods are only used within the Unity Editor.
- **Invoke methods**: Call these methods within Unity Editor scripts or editor windows to perform GUI drawing, data manipulation, and other editor-specific functionalities.

### For instance:

```
LanguageClassToolsEditor.DrawColoredButton("Click Me", 14f, 100f, Color.blue, Color.white, () =>
Debug.Log("Button Clicked!"));
// ... and use other methods as needed within the Unity Editor environment.
```

These methods are specifically tailored to enhance the editor experience by providing functionalities like drawing buttons, managing data lists, handling boolean extraction, displaying component images, and more within the Unity Editor environment.

## TextAlignmentConverter and TMP\_TextAlignmentConverter

The `TextAlignmentConverter` is a simple utility class in Unity that provides a method for converting horizontal text alignment (as represented by the `TextAnchor` enum) into a corresponding vertical alignment. Here's a breakdown of the class and method:

1. `TextAlignmentConverter` is a static class, which means it doesn't require an instance to be created. It's a collection of utility methods for text alignment conversion.
2. `GetVerticalAlignmentFromTextAlignment(TextAnchor alignment)` is a method that takes a `TextAnchor` value (representing horizontal alignment) as an input parameter and returns a string representing the corresponding vertical alignment.
3. The method uses a C# `switch` statement with pattern matching to determine the vertical alignment based on the provided horizontal alignment.
  - If the horizontal alignment is one of the upper values (`TextAnchor.UpperLeft`, `TextAnchor.UpperCenter`, or `TextAnchor.UpperRight`), it returns "Upper."
  - If the horizontal alignment is one of the middle values (`TextAnchor.MiddleLeft`, `TextAnchor.MiddleCenter`, or `TextAnchor.MiddleRight`), it returns "Middle."
  - If the horizontal alignment is one of the lower values (`TextAnchor.LowerLeft`, `TextAnchor.LowerCenter`, or `TextAnchor.LowerRight`), it returns "Lower."
  - If none of the expected cases match, it returns "Unknown."

This utility can be helpful when you need to convert horizontal text alignment settings into corresponding vertical alignment settings in your Unity application. It's a straightforward way to ensure consistency in how text alignment is handled, especially if you're working with text layout and UI elements.

# Components

## LanguageScript

This script, named **`LanguageScript`**, is designed to handle language localization within Unity. It includes an editor script for customizing the inspector view and an editor window for managing language save IDs. Let's break down the main elements:

### LanguageScript Class:

#### 1. Variables:

- **\_object (GameObject)**: The GameObject to which the script with the localized variable belongs.
- **textToSave (string)**: Holds the text that will be saved or used for localization purposes.
- **scriptName (string)**: The name of the script attached to the GameObject.
- **variableName (string)**: The name of the variable in the script that will hold the localized text.
- **updateMethod (bool)**: If true, the method specified by **`methodName`** will be called after updating the variable.
- **methodName (string)**: The name of the method to be called after updating the variable.
- **ID (float)**: The identifier for the text you want to display.
- **standardFile (string)**: The default language file to use if no save data is available.
- **selectedFile (string)**: The path to the currently selected language file.
- **savePath (string)**: The path to the JSON file where the language selection will be saved.
- **defaultFile (string)**: The path to the default language file based on the **`standardFile`** setting.
- **jsonNameInUnity (string)**: Path to the JSON file in Unity Editor.
- **folderNameInUnity (string)**: Folder containing language files in Unity Editor.
- **jsonNameInBuild (string)**: Path to the JSON file in the build.
- **folderNameInBuild (string)**: Folder containing language files in the build.

#### 2. Methods:

- **OnEnable()**: Called when the GameObject is enabled. Calls **`LanguageUpdate()`** to update the language.
- **LanguageUpdate()**: Updates the language based on the current settings and saved data.
- **ReadFile(string filePath)**: Reads and processes the language file specified by **`filePath`**.
- **ProcessLanguageOption(string line)**: Processes a language option defined in the given line.

### LanguageScriptEditor Class (Editor Script):

#### 1. Variables:

- **IDs (List<float>)**: A static list to hold IDs for language options.
- **savedID (bool)**: Flag to track if the language text option ID has been saved.

#### 2. Methods:

- **OnEnable()**: Loads data from a file into the **`IDs`** list.
- **OnInspectorGUI()**: Custom Inspector GUI for **`LanguageScript`**.
- **AddLanguageSaveID(LanguageScript script)**: Adds a language save ID, updates properties, saves data, and reloads IDs list.

### Usage:

#### 1. Inspector View:

- The script provides a customized inspector view for easy configuration.
- Allows you to set various parameters, including the script to localize, variables, language file paths, and more.
- Features a button to save settings, which invokes the Language Save Editor window.

#### 2. Localization:

- The script reads language files, identifies the specified language option ID, and updates the associated script's variable with the localized text.
- Optionally calls a method in the script after updating the variable.

#### 3. Editor Script:

- The **`LanguageScriptEditor`** class enhances the editor experience, allowing for easier customization and management of language save IDs.
- It provides an interface for saving settings, handling confirmation dialogs, and integrating with the Language Save Editor window.

Overall, this script facilitates language localization in Unity, providing tools to manage and update localized text in the editor environment and during runtime.

**Warning, this script uses the reflection system, which can have performance implications, so using it with caution is essential, especially if this script is called frequently or at critical performance moments.**

## LanguageCreateFile

The script named **LanguageCreateFile** is designed to facilitate the creation of language files within the Unity environment. It includes an editor script for customizing the inspector view and managing language save IDs. Additionally, it defines a data structure named **LanguageLines** to hold information about localized text lines. Let's break down the main elements:

### LanguageCreateFile Class:

#### 1. Variables:

- **languageLines (List<LanguageLines>)**: A list of **LanguageLines** objects that hold localized text data. Each object includes the localized text, an ID, a line index for file creation, and a flag to track whether the ID was found.
- **standardFile (string)**: The default language file to use if no save data is available.
- **selectedFile (string)**: The path to the currently selected language file.
- **savePath (string)**: The path to the JSON file where the language selection will be saved.
- **defaultFile (string)**: The path to the default language file based on the **standardFile** setting.
- **jsonNameInUnity (string)**: Path to the JSON file in Unity Editor.
- **folderNameInUnity (string)**: Folder containing language files in Unity Editor.
- **jsonNameInBuild (string)**: Path to the JSON file in the build.
- **folderNameInBuild (string)**: Folder containing language files in the build.
- **folderCreateNameInUnity (string)**: The folder name to save the created file in Unity Editor.
- **folderCreateNameInBuild (string)**: The folder name to save the created file in the build.
- **fileName (string)**: The name of the file to be created.
- **fileExtension (string)**: The extension of the file to be created.
- **fileLines (List<string>)**: The list of lines for the file to be created.

#### 2. Methods:

- **OnEnable()**: Called when the GameObject is enabled. Calls **LanguageUpdate()** to update the language.
- **LanguageUpdate()**: Updates the language based on the current settings and saved data.
- **ReadFile(string filePath)**: Reads and processes the language file specified by **filePath**.
- **ProcessLanguageOption(string line)**: Processes a language option defined in the given line.
- **CreateFile()**: Creates a language file based on the provided settings and localized text.

### LanguageCreateFileEditor Class (Editor Script):

#### 1. Variables:

- **IDs (List<float>)**: A static list to store IDs for language lines.
- **savedID (bool)**: Flag to track if the language line ID has been saved.

#### 2. Methods:

- **OnEnable()**: Loads data from a file into the **IDs** list.
- **OnInspectorGUI()**: Custom Inspector GUI for **LanguageCreateFile**.
- **AddLanguageSaveID(LanguageCreateFile script)**: Adds language save IDs, updates properties, saves data, and reloads IDs list.

### Usage:

#### 1. Inspector View:

- The script provides a customized inspector view for easy configuration.
- Allows you to set various parameters, including language lines, file paths, file creation settings, and more.
- Features a button to save settings, which invokes the Language Save Editor window.

#### 2. Localization:

- The script reads language files, identifies the specified language option ID, and updates the associated **LanguageLines** object with the localized text.
- The **ProcessLanguageOption** method updates the **LanguageLines** objects in the **languageLines** list.



### 3. File Creation:

- The script creates language files based on the provided settings.
- The ``CreateFile`` method creates the file, updates the lines with localized text, and writes the lines to the file.

### 4. Editor Script:

- The ``LanguageCreateFileEditor`` class enhances the editor experience, allowing for easier customization and management of language save IDs.
- It provides an interface for saving settings, handling confirmation dialogs, and integrating with the Language Save Editor window.

Overall, this script streamlines the process of creating language files within Unity, providing tools to manage and update localized text in the editor environment.

**Warning this component can be used for malicious purposes depending on the file format you define in its settings. It is your responsibility not to leave formats that can be used in malicious ways.**

## LanguageDropdown and TMP\_LanguageDropdown

This script is a part of a Unity project and is designed to manage a language dropdown menu in the Unity Editor. The script contains functionality to handle the selection of different language options, adjust UI elements based on the selected language, and save and load settings related to the language dropdown.

Let's break down the key components of the script:

### Variables:

- **languageDropdown:** A reference to the Unity UI Dropdown component used to display language options.
- **itemText and captionText:** References to Text components used for displaying the selected item and caption, respectively.
- **options:** A list of LanguageOptions objects representing different language options with associated text, sprites, and IDs.
- **ID:** An ID to identify the language options.
- **adjustRectList:** A list of adjustments for UI elements.
- **font:** The index of the font to be used for displaying the text.
- **rotation, localScale, anchorMin, anchorMax, anchoredPosition, sizeDelta, pivot:** Various parameters for adjusting the position, rotation, and scale of UI elements based on the selected language.
- **alignment:** The alignment of the text.
- **reverseText:** A flag to determine whether to reverse the text.
- **standardFile, selectedFile, savePath, defaultFile:** File paths and names for storing and loading language-related information.
- **jsonNameInUnity, folderNameInUnity, jsonNameInBuild, folderNameInBuild:** Paths for language-related files in Unity Editor and the build.
- **wasImported:** A flag to track whether settings were imported.

### Methods:

- **OnEnable():** Initializes the script, retrieves necessary references, and calls `LanguageUpdate()`.
- **LanguageUpdate():** Updates the language options based on saved settings or defaults.
- **ReadFile(string filePath):** Reads and processes a language file, updating language options.
- **ProcessOption():** Clears and populates the dropdown with updated language options.
- **ProcessLanguageOption(string line):** Processes individual language options from a file.
- **SetTextAlignment():** Sets the text alignment based on the specified alignment index.
- **SaveDropdown():** Saves the settings related to the language dropdown.

### Editor-Only Section:

The script includes a section enclosed in `#if UNITY_EDITOR` and `#endif`. This part is meant for the Unity Editor and includes an editor script (`LanguageDropdownEditor`) to manage the settings in the Unity Editor interface. It allows for importing and saving settings, displaying warnings for duplicate IDs, and exporting IDs to an AdjustRect system.

### Explanation of Editor Script:

- **OnInspectorGUI():** Handles the inspector GUI for the LanguageDropdown component, providing buttons for importing and saving settings, displaying warnings, and adjusting various properties.
- **AddLanguageSaveID(LanguageDropdown script):** Adds or updates language save IDs based on the settings of the LanguageDropdown component.

### Usage:

1. Attach the `LanguageDropdown` script to a GameObject in the Unity Editor.
2. Assign the required references such as the Dropdown, itemText, and captionText.
3. Populate the `options` list with different language options.
4. Set up font, rotation, scale, anchor positions, etc., for each language option.
5. Adjust default language settings and file paths.
6. Use the Unity Editor interface to import, save, and adjust settings for the language dropdown.

The script is designed to provide a flexible language selection system in Unity, allowing for customization of various text and UI-related properties based on the selected language.

## LanguageDropdownOptions and TMP\_LanguageDropdownOptions

This script is designed to manage language selection within a Unity project. Here's a breakdown of its functionality:

### Variables:

- **languageDropdown:** Dropdown UI component displaying language options.
- **languageName:** Default language to use if no saved data is available.
- **selectedLanguage, systemLanguage, savePath, selectedFile, fileSavePath:** Variables to store information about the selected language, system language, paths for saving language data, and the selected language file.
- **jsonNameInUnity, jsonSaveNameInUnity, folderNameInUnity:** Paths and folder names specific to the Unity Editor.
- **jsonNameInBuild, jsonSaveNameInBuild, folderNameInBuild:** Paths and folder names for the build.

### Methods:

- **Start():** Triggered at the script's start.
  - Determines the system language, sets file paths based on the environment (Unity Editor or build).
  - Loads available language files.
  - Checks for saved language data. If none exists, it tries to match the system language with available language files.
  - Sets default language if no saved data or system language match is found.
  - Adds a listener to handle language selection changes.
- **OnLanguageChanged():** Called when the language selection changes.
  - Updates the selected language, finds the corresponding file, and saves it.
  - Triggers the language update to apply the selected language across the project.
- **LanguageUpdate():** Searches for specific components that need a language update and triggers their `LanguageUpdate()` method.
  - Iterates through various scripts/components for language update.
  - Specifically targets different types of UI elements, audio players, and other elements that display text or require language-specific adjustments.
- **OnFileChanged():** Saves the selected file to JSON when the file selection changes.

### Usage:

#### 1. Initialization (Start()):

- Determines the system language and available language files.
- Loads saved language data if available.
- Sets default language if no saved data or match with the system language is found.

#### 2. Language Selection:

- Changes trigger `OnLanguageChanged()`.
- Updates the selected language, finds the corresponding file, and applies the new language settings across the project.

#### 3. File Selection:

- Changing the file triggers `OnFileChanged()`.
- Saves the selected file to JSON.

#### 4. Language Update:

- `LanguageUpdate()` method triggers updates across different UI components and elements that display language-specific content.

This script essentially manages language selection, ensures the correct language files are loaded and applied, and updates various UI elements to reflect the selected language.

## LanguageText and TMP\_LanguageText

This script in Unity is designed to handle language localization for UI Text components. It allows for adjusting various properties of the Text component based on the selected language. Let's break down the key components of the script:

### LanguageText Class

#### - Variables:

- **`languageText`**: Reference to the Text component for displaying the localized text.
- **`ID`**: ID to identify the specific language text option.
- **`adjustRectList`**: List of adjustments for UI elements.
- **`translateText`**: Flag to enable/disable text translation.
- Various properties for text appearance (fontSize, font, rotation, localScale, anchorMin, anchorMax, etc.).
- **`standardFile`**: The default language file to use if no save data is available.
- **`selectedFile`**, **`savePath`**, **`defaultFile`**: Paths to the selected language file, save path, and default language file.
- **`jsonNameInUnity`**, **`folderNameInUnity`**: Paths for JSON file and folder in Unity Editor.
- **`jsonNameInBuild`**, **`folderNameInBuild`**: Paths for JSON file and folder in the build.
- **`wasImported`**: Flag to track whether settings were imported.

#### - Methods:

- **`OnEnable()`**: Called when the component is enabled. Calls **`LanguageUpdate()`** to update the language text.
- **`LanguageUpdate()`**: Updates the language text based on the selected language file.
- **`ReadFile(string filePath)`**: Reads and processes the language file specified by **`filePath`**.
- **`ProcessLanguageOption(string line)`**: Processes a language option defined in the given line.
- **`SetTextAlignment()`**: Sets the alignment of the language text component.
- **`SaveText()`**: Saves text properties like rotation, scale, anchor positions, and sizes.

### LanguageTextEditor Class

#### - Custom Editor for LanguageText:

- Custom editor for the **`LanguageText`** component in Unity Editor.
- Handles importing and saving language settings.
- Implements **`OnInspectorGUI()`** to provide a custom inspector interface.

### Explanation of the Usage:

- Attach the **`LanguageText`** script to any GameObject with a Text component.
- Assign the **`languageText`** field to the Text component you want to localize.
- Set the **`ID`** to identify the specific language text option.
- Adjust other properties (font size, font, rotation, etc.) as needed.
- Define language options in external files with specific formats.
- Use the custom editor to import and save language settings.
- The script automatically updates the text based on the selected language or default language.

This script enables easy localization of Unity UI text elements, and the custom editor streamlines the process of importing and saving language settings.

## LanguageTextInputField and TMP\_LanguageTextInputField

This script IS part of a Unity project and is designed to manage language localization for UI elements. Let's break down the script into its key components:

### Variables:

- **`textComponent`**: Reference to the Text component for displaying the localized text.
- **`placeholder`**: Reference to a **`LanguageText`** object.
- **`ID`**: ID to identify the specific language text option.
- **`adjustRectList`**: List of adjustments for UI elements.
- **`fontSize`**, **`font`**: Font size and font index for displaying the text.
- **`rotation`**, **`localScale`**: Rotation and scale of the text.
- **`anchorMin`**, **`anchorMax`**, **`anchoredPosition`**, **`sizeDelta`**, **`pivot`**: Various RectTransform properties.
- **`alignment`**: The alignment of the text.
- **`standardFile`**: Default language file to use if no save data is available.
- **`selectedFile`**: The path to the currently selected language file.
- **`savePath`**: Path to the JSON file where the selected language will be saved.
- **`defaultFile`**: Path to the default language file.
- **`jsonNameInUnity`**, **`folderNameInUnity`**: Paths for language files in Unity Editor.
- **`jsonNameInBuild`**, **`folderNameInBuild`**: Paths for language files in the build.
- **`wasImported`**: Flag to track whether settings were imported.
- **`foundID`**: Flag to track if the language text option ID has been found.
- **`isQuaternionFoldout`**: Flag to control quaternion foldout in the inspector.

### Methods:

- **`OnEnable`**: Called when the component is enabled. Invokes **`LanguageUpdate`**.
- **`LanguageUpdate`**: Updates the language text based on the selected or default language.
- **`ReadFile`**: Reads and processes the language file.
- **`ProcessLanguageOption`**: Processes a language option defined in a given line.
- **`SetTextAlignment`**: Sets the alignment of the language text component.
- **`SaveInputField`**: Saves input field properties like rotation, scale, etc., into corresponding variables.

### Unity Editor Custom Editor:

- **`LanguageTextInputFieldEditor`**: Custom editor class for the **`LanguageTextInputField`**. It provides an editor interface with buttons to import and save settings, inspector fields, and a method (**`AddLanguageSaveID`**) to update language save IDs in the editor.

### Usage:

1. Attach this script to a UI Text component in Unity.
2. Set the necessary properties such as font, size, alignment, etc.
3. Assign a **`LanguageText`** object to the **`placeholder`** field.
4. Optionally, configure language file paths, default language, and other settings.
5. Use the provided editor buttons to import and save settings.
6. Adjust RectTransform properties and other text-related settings.

Note: The script essentially facilitates the management and localization of text input fields in Unity, providing tools to configure and update text properties for different languages.

## LanguageTextMesh and TMP\_LanguageTextMesh

This script is a Unity MonoBehaviour that facilitates the localization of TextMesh components. It allows you to associate different language settings with a TextMesh component, and it includes an editor script for managing these settings within the Unity Editor. Let's break down the script:

### `LanguageTextMesh` Class:

#### - Fields:

- **`languageTextMesh`**: Reference to the attached TextMesh component for displaying localized text.
- **`ID`**: The identifier for the text you want to display.
- **`fontSize`**: The font size to apply to the TextMesh component.
- **`font`**: The index of the font to apply to the TextMesh component.
- **`fontListObject`**: Scriptable object containing a list of fonts for TextMesh.
- **`reverseText`**: If true, the text will be reversed before displaying.
- **`standardFile`**: The default language file to use if no save data is available.
- **`selectedFile`**: The path to the currently selected language file.
- **`savePath`**: The path to the JSON file where the language selection will be saved.
- **`defaultFile`**: The path to the default language file based on the **`standardFile`** setting.
- **`jsonNameInUnity`**, **`folderNameInUnity`**: Paths for the JSON file and folder in Unity Editor.
- **`jsonNameInBuild`**, **`folderNameInBuild`**: Paths for the JSON file and folder in the build.
- **`wasImported`**: Flag to track whether settings were imported.
- **`foundID`**: Flag to track if the language text option ID has been found.

#### - Methods:

- **`OnEnable()`**: Called when the GameObject is enabled. Invokes **`LanguageUpdate()`** to update the language settings.
- **`LanguageUpdate()`**: Updates the language settings by reading the JSON file or using default settings.
- **`ReadFile(string filePath)`**: Reads and processes the language file specified by **`filePath`**.
- **`ProcessLanguageOption(string line)`**: Processes a language option defined in the given line.
- **`SavaTextMesh()`**: Saves the current TextMesh settings to the script fields.

### `LanguageTextMeshEditor` Class:

#### - Editor Script:

- Provides a custom editor for the **`LanguageTextMesh`** component in the Unity Editor.
- Handles importing and saving settings, displaying property fields, and managing language save IDs.
- Utilizes a custom window (**`LanguageSaveEditorWindow`**) for managing language save IDs.

### Usage:

- Attach the **`LanguageTextMesh`** script to a GameObject with a TextMesh component.
- Set the desired language settings in the Unity Editor.
- Use the editor buttons ("Import Settings" and "Save Settings") to manage and save language settings.
- The script automatically updates the TextMesh component based on the selected language.

Note: The script essentially manages the localization of TextMesh components based on language files, allowing easy modification and updating of text properties within the Unity Editor.

## LanguageImage

This script is a Unity MonoBehaviour that facilitates the localization of images based on the selected language. It allows you to associate different images with specific languages and includes functionality to load and set these images dynamically. Let's break down the script:

### `LanguageImage` Class:

#### - Fields:

- **`image`**: Reference to the Image component used to display the language-specific image.
- **`imageName`**: The name of the image file to be displayed.
- **`languageName`**: The name of the language.
- **`standardFile`**: The default language file to use if no save data is available.
- **`selectedFile`**: The path to the currently selected language file.
- **`savePath`**: The path to the JSON file where the selected language will be saved.
- **`defaultFile`**: The path to the default language file.
- **`jsonNameInUnity`**, **`folderNameInUnity`**: Paths for the JSON file and folder in Unity Editor.
- **`jsonNameInBuild`**, **`folderNameInBuild`**: Paths for the JSON file and folder in the build.

#### - Methods:

- **`OnEnable()`**: Called when the component is enabled. Invokes **`LanguageUpdate()`** to update the Language Image.
- **`LanguageUpdate()`**: Initiates the coroutine **`LoadAndSetImage()`** to load and set the image based on the selected language.
- **`LoadAndSetImage()`**: Coroutine responsible for loading and setting the image based on language and platform.

### Usage:

1. Attach the **`LanguageImage`** script to a GameObject with an Image component.
2. Set the desired language settings and image file name in the Unity Editor.
3. Use the editor fields to specify the standard language file, image file name, and file paths.
4. The script will automatically load and set the image based on the selected language when the GameObject is enabled.

### How it Works:

#### 1. Initialization:

- The script initializes its settings and paths based on the Unity Editor or build environment.
- It identifies the save path for language selection and finds the default language file.

#### 2. Language Update:

- When the GameObject is enabled, **`LanguageUpdate()`** is called, initiating the coroutine **`LoadAndSetImage()`**.

#### 3. Image Loading:

- The script constructs the file path for the image based on the language, platform, and specified file name.
- It uses a UnityWebRequest to load the image file asynchronously.

#### 4. Image Setting:

- If the image loading is successful, it creates a Sprite from the loaded texture and sets it as the sprite for the Image component.
- If there is an error loading the image, it logs an error message to the console.

### Note:

- The script assumes that images are organized within language-specific folders, and the file structure follows a specific pattern.
- Make sure that the **`DownloadHandlerTexture`** namespace is imported or available in your project.
- It uses Unity's **`UnityWebRequest`** to load images, so make sure that UnityWebRequest is supported in the target platform.



Ensure that the image files and language folders are organized according to the specified structure for this script to work correctly.

## LanguageRawImage

This Unity script, **LanguageRawImage**, is designed to facilitate the localization of RawImages in a Unity project. It allows you to associate different images with specific languages and dynamically load and set the RawImage texture based on the selected language. Here's an explanation of the script:

### **LanguageRawImage` Class:**

#### **- Fields:**

- **rawImage`**: Reference to the RawImage component used to display the language-specific image.
- **imageFileName`**: The name of the image file to be displayed.
- **languageName`**: The name of the language.
- **standardFile`**: The default language file to use if no save data is available.
- **selectedFile`**: The path to the currently selected language file.
- **savePath`**: The path to the JSON file where the selected language will be saved.
- **defaultFile`**: The path to the default language file.
- **jsonNameInUnity`, `folderNameInUnity`**: Paths for the JSON file and folder in Unity Editor.
- **jsonNameInBuild`, `folderNameInBuild`**: Paths for the JSON file and folder in the build.

#### **- Methods:**

- **OnEnable()**: Called when the component is enabled. Invokes **LanguageUpdate()** to update the Language RawImage.
- **LanguageUpdate()**: Initiates the coroutine **LoadAndSetImage()** to load and set the image based on the selected language.
- **LoadAndSetImage()**: Coroutine responsible for loading and setting the RawImage texture based on language and platform.

### **Usage:**

1. Attach the **LanguageRawImage`** script to a GameObject with a RawImage component.
2. Set the desired language settings and image file name in the Unity Editor.
3. Use the editor fields to specify the standard language file, image file name, and file paths.
4. The script will automatically load and set the RawImage texture based on the selected language when the GameObject is enabled.

### **How it Works:**

#### **1. Initialization:**

- The script initializes its settings and paths based on the Unity Editor or build environment.
- It identifies the save path for language selection and finds the default language file.

#### **2. Language Update:**

- When the GameObject is enabled, **LanguageUpdate()** is called, initiating the coroutine **LoadAndSetImage()**.

#### **3. Image Loading:**

- The script constructs the file path for the image based on the language, platform, and specified file name.
- It uses a UnityWebRequest to load the image file asynchronously.

#### **4. Image Setting:**

- If the image loading is successful, it sets the RawImage texture to the loaded texture.
- If there is an error loading the image, it logs an error message to the console.

### **Note:**

- The script assumes that images are organized within language-specific folders, and the file structure follows a specific pattern.
- Make sure that the **DownloadHandlerTexture`** namespace is imported or available in your project.
- It uses Unity's **UnityWebRequest`** to load images, so make sure that UnityWebRequest is supported in the target platform.

Ensure that the image files and language folders are organized according to the specified structure for this script to work correctly.

## AdjustRect

This Unity script, named **AdjustRect**, is designed to allow for the adjustment of RectTransform settings based on language files, with associated editor functionality. Let's break down its structure, variables, methods, and usage:

### **AdjustRect** Class:

#### Fields:

##### - Settings:

- **rectTransform**: Reference to the RectTransform that the script will adjust.
- **ID**: An identifier for the language text option.

##### - Rect Transform Settings:

- **rotation**, **localScale**, **anchorMin**, **anchorMax**, **anchoredPosition**, **sizeDelta**, **pivot**:

Various RectTransform settings to be adjusted.

##### - Default language if there is no save:

- **standardFile**: The default language file to use if no save data is available.

##### - Automatic Information:

- **selectedFile**: The path to the currently selected language file.
- **savePath**: The path to the JSON file where the selected language will be saved.
- **defaultFile**: The path to the default language file.

##### - Archives Location:

- Paths and folders related to language files in Unity Editor and builds.

##### - **wasImported**: A flag to track whether settings were imported.

#### Methods:

- **OnEnable()**: Called when the component is enabled. Invokes **LanguageUpdate()** to update the language text.
- **LanguageUpdate()**: Updates the language text based on the selected language file.
- **ReadFile(string filePath)**: Reads and processes a language file specified by **filePath**.
- **ProcessLanguageOption(string line)**: Processes a language option defined in a given line from the language file.
- **SaveRectTransform()**: Saves RectTransform settings to the **AdjustRect** instance.

#### Usage:

1. Attach the **AdjustRect** script to a GameObject with a RectTransform.
2. Set the desired language settings, including the default language file and paths for language files in the Unity Editor and builds.
3. Use the editor fields to specify RectTransform settings and the associated language ID.
4. The script will automatically load and adjust RectTransform settings based on the selected language when the GameObject is enabled.
5. In the Unity Editor, additional editor functionality is provided through the custom editor **AdjustRectEditor**, allowing the import and export of RectTransform settings.

### **AdjustRectEditor** Class (Editor):

This nested class handles the custom editor functionality for the **AdjustRect** script in the Unity Editor. It includes methods to import and save settings, display a custom inspector GUI, and manage language save IDs.

### **AdjustRectList** Class:

A serializable class to hold a list of **AdjustRect** objects with associated settings. This could be useful for managing multiple **AdjustRect** instances and their settings.

#### Notes:

- The script appears to load language files and adjust RectTransform settings based on language-specific IDs.
- The custom editor provides a UI for importing and saving RectTransform settings, displaying warnings for existing IDs.
- Ensure that the associated **LanguageClassTools** and **LanguageSaveEditorWindow** classes are available in the project.

The script essentially serves as a utility to manage and apply different RectTransform settings based on language-specific files or IDs within the Unity Editor environment.

## LanguageAudioPlayer

This Unity script, named **LanguageAudioPlayer**, is designed to play audio files based on language selection. Let's break down its structure, variables, methods, and usage:

### **LanguageAudioPlayer** Class:

#### **Fields:**

##### **- Settings:**

- **audioSource**: Reference to the AudioSource component used to play the audio.
- **fileName**: The name of the audio file to be played.

##### **- Archives Location:**

- **languageName**: The name of the language.

##### **- Default language if there is no save:**

- **standardFile**: The default language file to use if no save data is available.

##### **- Automatic Information:**

- **selectedFile**: The path to the currently selected language file.
- **savePath**: The path to the JSON file where the selected language will be saved.
- **defaultFile**: The path to the default language file.

##### **- Archives Location:**

- Paths and folders related to language files in Unity Editor and builds.

#### **Methods:**

- **OnEnable()**: Called when the component is enabled. Invokes **LanguageUpdate()** to update the language audio player.
- **LanguageUpdate()**: Updates the language audio player based on the selected language file.
- **LoadAndPlayAudio()**: Coroutine method to load and play the audio file.

#### **Usage:**

1. Attach the **LanguageAudioPlayer** script to a GameObject with an AudioSource component.
2. Set the desired language settings, including the default language file, file name, and paths for language files in the Unity Editor and builds.
3. Use the editor fields to specify the audio file name and associated language.
4. The script will automatically load and play the audio file based on the selected language when the GameObject is enabled.
5. Ensure that the associated **LanguageClassTools** class is available in the project.

#### **Notes:**

- The script loads language files and plays audio based on language selection.
- The **LanguageUpdate()** method checks for the existence of the save file and loads the selected language or falls back to the default language.
- The audio file path is constructed based on the language and platform.
- The script uses a Unity **UnityWebRequest** to load the audio file asynchronously, and on success, assigns the audio clip to the AudioSource and plays it.

This script is suitable for scenarios where you need to play different audio files based on the selected language in a Unity application. It ensures that the appropriate language-specific audio is played when the associated GameObject is enabled.

## AdjustSizeToDropdown and TMP\_AdjustSizeToDropdown

This script, **AdjustSizeToDropdown**, dynamically adjusts the size of a UI element, typically used for dropdowns, based on the content it holds. Let's go through its components:

### Variables:

- **Manual Size Adjustment:** Allows manual adjustment of the UI element size.
- **Size Multiplier:** Multiplier used for dynamically adjusting the size.
- **Margin:** Extra spacing between the UI element and canvas boundaries.
- **Fit Direction:** Enum defining whether to fit the element to the left or right.
- **Parent Rect, Canvas Rect:** References to the RectTransform of the UI element and the Canvas.
- **Canvas Width, Object Width:** Width of the canvas and the UI element.
- **TextList:** List of **BrokenTexts** containing Text components and their line break status.
- **TextIsBroken:** Flag indicating if any text breaks into multiple lines.
- **Size Adjustment:** Value used to adjust the UI element size.

### Methods:

- **Start():** Fetches references to the UI element's RectTransform and the Canvas, initializes the list of Text components, and sets overflow mode.
- **FixedUpdate():** Calculates and adjusts the UI element's size based on text content, fit direction, and canvas boundaries.
- **CheckTheLineBreak():** Checks if any Text component breaks into multiple lines.

### Usage:

- Attach this script to a GameObject containing a UI element (dropdown).
- Set the fit direction and other parameters in the Inspector.
- Ensure the UI element has Text components inside it for accurate size adjustment based on content.
- Adjustments are made dynamically during runtime based on the text content and canvas boundaries.

### Important Notes:

- The script dynamically resizes UI elements based on text content and canvas boundaries.
- It checks if any Text component inside the UI element breaks into multiple lines.
- The adjustment occurs by changing the offsetMin and offsetMax of the RectTransform, considering the chosen fit direction.
- Adjustments take place in **FixedUpdate()** to ensure it reacts consistently regardless of frame rate.

Overall, this script is useful for ensuring that dropdowns or similar UI elements adjust their size dynamically based on the content they hold while considering canvas boundaries.

## LanguageFontListData and TMP\_LanguageFontListData

This script, `LanguageFontListData`, is a `ScriptableObject` used to hold a list of Font assets. Let's break it down:

### Variables:

- **FontList:** A list of Font assets used for the `LanguageText` script or any other scripts requiring Font assets.

### CreateAssetMenu:

- **FileName:** Specifies the default file name when creating an instance of this `ScriptableObject`.
- **MenuName:** Defines the path in the Editor's **"Create"** menu where you can create instances of this `ScriptableObject`.

### Usage:

- Create instances of this `ScriptableObject` asset in the Unity Editor by right-clicking in the Assets folder, going to **"Create,"** and selecting **"Language/Language Font List Data."**
- Once created, you can populate the `fontList` field by dragging and dropping Font assets into the list from your project folders.
- Use this asset as a centralized repository for Font assets, especially when multiple scripts or components need access to various fonts.
- Access this asset from other scripts by referencing it directly or via serialized fields in `MonoBehaviours`.

### Purpose:

- Provides a convenient way to manage and store Font assets as a collection in a centralized `ScriptableObject`.
- Offers a cleaner method than hardcoding or referencing individual Font assets in multiple places throughout the project.
- Allows easy modification of Font lists without changing scripts directly.

### Summary:

This `ScriptableObject` serves as a container for a list of Font assets. It streamlines the management of Font assets by providing a centralized and easily modifiable collection. Other scripts can reference this `ScriptableObject` to access the Font assets listed within it, offering a cleaner and more organized approach to handling fonts across a Unity project.

## LanguageSettingsData

This script, `LanguageSettingsData`, is a ScriptableObject designed to hold various settings and references related to language localization. Let's break down its components:

### Variables:

- **fontListData**: A reference to a `LanguageFontListData` scriptable object that holds a list of fonts for regular text.
- **TMP\_fontListData**: A reference to a `TMP_LanguageFontListData` scriptable object that holds a list of fonts for TextMeshPro (TMP) text.
- **jsonNameInUnity**: The path to the JSON file for language selection in the Unity Editor.
- **jsonSaveNameInUnity**: The path to the JSON file for the selected language file in the Unity Editor.
- **folderNameInUnity**: The folder containing language files in the Unity Editor.
- **jsonNameInBuild**: The path to the JSON file for language selection in the build.
- **jsonSaveNameInBuild**: The path to the JSON file for the selected language file in the build.
- **folderNameInBuild**: The folder containing language files in the build.

### Usage:

- This script is used as a configuration data holder for language settings within the Unity Editor and the built application.
- It references other scriptable objects (`LanguageFontListData` and `TMP_LanguageFontListData`) that store lists of fonts for regular text and TextMeshPro text, respectively.
- The paths and folder locations are used to specify where the language-related JSON files and language files are stored.

### Purpose:

- Provides a centralized place to store and manage various language-related settings for the application.
- Facilitates the use of different font lists for regular text and TextMeshPro text, allowing flexibility in font selection for different text components.
- Specifies the locations where language-related JSON files are stored, both in the Unity Editor and the built application.

### Summary:

`LanguageSettingsData` acts as a container for language settings, referencing other scriptable objects for font lists and specifying file paths and folder locations for language-related JSON files. It simplifies the management of language-related configurations and settings, providing a clean and organized approach to handle various aspects of language localization within a Unity project.



## LanguageInitialization

This script, ``LanguageInitialization``, handles the initialization of language-related settings, loads fonts from asset bundles, saves and loads font lists, and manages language file selections. Let's break it down:

### Variables:

- **languageSettingsData**: Reference to ``LanguageSettingsData`` scriptable object containing language settings.
- **selectedLanguage** and **systemLanguage**: Strings to store the selected and system languages.
- **savePath** and **selectedFile**: File paths for saving language selections.
- **fileSavePath**: Path for saving language file selections.
- **fontListData** and **TMP\_fontListData**: References to font list data for regular fonts and TextMeshPro fonts.
- **jsonNameInUnity**, **jsonSaveNameInUnity**, **folderNameInUnity**: File paths and folder names in the Unity Editor.
- **jsonNameInBuild**, **jsonSaveNameInBuild**, **folderNameInBuild**: File paths and folder names in a build.

### Methods:

- **RunGameInitialization()**: Entry point method called during game initialization. Loads settings data and initializes language settings.
- **LoadSettingsData()**: Loads language settings from a Unity asset and assigns them to variables.
- **LanguageStart()**: Initiates language setup, detects system language, and selects appropriate language files if none exist.
- **LoadFontsFromAssetBundle()** and **TMP\_LoadFontsFromAssetBundle()**: Load fonts and TextMeshPro fonts respectively from asset bundles in the "Font" directory.
- **SaveFontLists()** and **LoadFontLists()**: Save and load font lists (regular fonts and TextMeshPro fonts) to/from text files.

### Usage:

- **Initialization**: Executed during the game's start. Loads language settings data and initializes language settings.
- **Font Loading**: Loads fonts and TextMeshPro fonts from asset bundles if not in the Unity Editor.
- **Saving & Loading Fonts**: Saves font lists to text files and loads them during runtime.
- **Selecting Language**: Checks system language and selects an appropriate language file if none are saved.

### Purpose:

- Centralizes language initialization, font loading, and management of language file selections.
- Handles loading fonts, both regular and TextMeshPro, from asset bundles and stores them in respective font lists.
- Manages the selection of language files based on system language and saves/loads these selections.

### Summary:

``LanguageInitialization`` serves as a pivotal script for language setup, font loading, and language file selection management. It centralizes the handling of language settings, font loading from asset bundles, and saving/loading font lists from text files. The script ensures proper language initialization, font loading, and language file selection based on system language, providing flexibility in managing different language-related aspects within a Unity project.

# Tools

## SceneCreateObject

This script, **SceneCreateObject**, is an Editor script in Unity. It's responsible for creating specific game objects via menu items within the Unity Editor. Let's break it down:

### Variables:

- **prefabPath**: String containing the path to the prefabs used for instantiating new game objects.

### Methods:

- **CreateFileLT()**: Creates a new game object from a prefab named "Language Create File (LT)".
- **AudioSourceLT()**: Creates a new game object from a prefab named "Audio Source (LT)".

### Usage:

- **Menu Items**: These methods are tied to menu items in Unity's GameObject menu under different categories (**Language** and **Audio**). When you click on these menu items, Unity executes the corresponding method.
- **Prefab Loading**: Loads prefabs from specified paths using **AssetDatabase.LoadAssetAtPath**.
- **Game Object Instantiation**: Uses **PrefabUtility.InstantiatePrefab** to instantiate prefabs as new game objects.
- **Naming and Unpacking**: Renames the instantiated game objects and unpacks them from the prefab state.
- **Scene Marking**: Marks the scene as dirty to indicate that changes have been made.

### Purpose:

- **Prefab Instantiation**: Allows users to easily create specific game objects within the Unity Editor by selecting specific menu items.
- **Automated Actions**: Renames, unpacks prefabs, and marks scenes as dirty automatically after the creation of new game objects.

### Summary:

The **SceneCreateObject** script simplifies the creation of specific game objects by providing menu items within Unity's GameObject menu. It uses prefabs to instantiate new game objects and performs automatic actions like renaming, unpacking prefabs, and marking the scene as dirty to streamline the workflow for developers working within the Unity Editor.

## CanvasCreateUiImageObjects

This script, **CanvasCreateUiImageObjects**, is responsible for creating UI Images and Raw Images within a Canvas in the Unity Editor. Here's a breakdown:

### Variables:

- **prefabPath**: String containing the path to the prefabs used for instantiating new UI Images or Raw Images.

### Methods:

- **CreateImageLT()**: Creates a new UI Image.
- **CreateRawImageLT()**: Creates a new Raw Image.

### Functionality:

- **Canvas Check**: Looks for a Canvas object in the scene. If not found, creates a new Canvas with default settings and an EventSystem to handle user interaction.
- **Prefab Loading**: Loads the prefab for the desired UI element from a specified path using **AssetDatabase.LoadAssetAtPath**.
- **Prefab Instantiation**:
  - If an object is selected inside the Canvas, it creates a new prefab from the existing prefab as a child of the selected object.
  - If nothing is selected or the selection is not a child of the Canvas, it creates a new prefab from the existing prefab as a child of the Canvas itself.
- **Prefab Renaming**: Renames the created prefab to "Image (LT)" or "RawImage (LT)" respectively.
- **Prefab Unpacking**: Unpacks the instantiated prefab from the prefab state.
- **Scene Marking**: Marks the scene as dirty to indicate that changes have been made.

### Usage:

- **Menu Items**: These methods are menu items in the Unity Editor's GameObject > UI menu. When clicked, they execute the corresponding method, creating the specified UI elements.
- **Prefab Handling**: Loads and instantiates prefabs based on user selection within the Canvas hierarchy.
- **Automatic Actions**: Renames, unpacks prefabs, and marks scenes as dirty automatically after the creation of new UI elements.

### Purpose:

- **Simplified UI Element Creation**: Allows developers to create UI Images and Raw Images easily via menu items within the Unity Editor.
- **Streamlined Workflow**: Automates the process of creating UI elements within a Canvas, reducing manual steps.

### Summary:

The **CanvasCreateUiImageObjects** script simplifies the creation of UI Images and Raw Images within a Canvas in the Unity Editor. It provides menu items for easy access, loads prefabs, manages instantiation, and handles the creation of these UI elements within the Canvas hierarchy, facilitating the UI design process for developers.

## CanvasCreateUiObjects and TMP\_CanvasCreateUiObjects

This script, **CanvasCreateUiObjects**, is designed to create various UI elements within a Canvas in the Unity Editor. Let's break it down:

### Variables:

- **prefabPath**: Contains the path to the prefabs used for instantiating different UI elements.

### Methods:

- **CreateLanguageDropdownLT()**: Creates a Language Dropdown UI element.
- **Create2TextLT()**: Creates a Text UI element.
- **CreateToggleLT()**: Creates a Toggle UI element.
- **CreateButtonLT()**: Creates a Button UI element.
- **CreateDropdownLT()**: Creates a Dropdown UI element.
- **CreateInputFieldLT()**: Creates an InputField UI element.

### Functionality:

- **Canvas Check**: Looks for a Canvas object in the scene. If not found, it creates a new Canvas with default settings and an EventSystem to handle user interaction.
- **Prefab Loading**: Loads the prefab for the desired UI element from a specified path using **AssetDatabase.LoadAssetAtPath**.
- **Prefab Instantiation**:
  - If an object is selected inside the Canvas, it creates a new prefab from the existing prefab as a child of the selected object.
  - If nothing is selected or the selection is not a child of the Canvas, it creates a new prefab from the existing prefab as a child of the Canvas itself.
- **Prefab Renaming**: Renames the created prefab based on the UI element created.
- **Prefab Unpacking**: Unpacks the instantiated prefab from the prefab state.
- **Scene Marking**: Marks the scene as dirty to indicate that changes have been made.

### Usage:

- **Menu Items**: These methods are menu items in the Unity Editor's GameObject > UI > Legacy menu. When clicked, they execute the corresponding method, creating the specified UI elements.
- **Prefab Handling**: Loads and instantiates prefabs based on user selection within the Canvas hierarchy.
- **Automatic Actions**: Renames, unpacks prefabs, and marks scenes as dirty automatically after the creation of new UI elements.

### Purpose:

- **Streamlined UI Creation**: Provides a quick way to create various UI elements within a Canvas without manual setup.
- **Enhanced Workflow**: Automates the process of creating UI elements within the Canvas, reducing manual steps and enhancing efficiency during UI development.

### Summary:

The **CanvasCreateUiObjects** script simplifies the creation of multiple UI elements within a Canvas in the Unity Editor. It offers menu items for easy access, loads prefabs, manages instantiation, and handles the creation of these UI elements within the Canvas hierarchy, aiding in the UI design process for developers.

## SceneCreateTextObject and TMP\_SceneCreateTextObject

This script, named `SceneCreateTextObject`, is designed to create a new text object within a Unity scene. Let's break down its structure and functionality:

### Variables:

- **prefabPath**: A string containing the path to the prefab used for instantiating a new text object.

### Methods:

- **CreateNewTextLT()**: This method is called when the user selects the menu item "GameObject/3D Object/Legacy/New Text (LT Legacy)" in the Unity Editor.

### Functionality:

- **Prefab Path Definition**: Specifies the path to the prefab for the new text object. The prefab path is manually added and points to the location of the prefab in the project.
- **Prefab Loading**: Attempts to load the prefab from the specified path using `AssetDatabase.LoadAssetAtPath<GameObject>(prefabPath)`.
- **Prefab Instantiation**:
  - Checks if the original prefab was successfully loaded. If not, it logs an error and returns.
  - Gets the currently selected game object in the scene.
  - Creates a new game object, `newGameObject`, by instantiating the loaded prefab. If there is a selected game object, the new object is created as a child of the selected object.
- **Object Naming**: Renames the newly created game object to "New Text (LT Legacy)".
- **Prefab Unpacking**: Unpacks the created prefab instance using `PrefabUtility.UnpackPrefabInstance`. This operation removes the association with the original prefab and allows for individual modifications to the instance.
- **Scene Modification Indication**: Marks the active scene as dirty using `EditorSceneManager.MarkSceneDirty`. This step is necessary to indicate that changes have been made to the scene.

### Usage:

- **Menu Item**: The method `CreateNewTextLT` is associated with the menu item "GameObject/3D Object/Legacy/New Text (LT Legacy)". When a user selects this menu item, the `CreateNewTextLT` method is triggered.

### Purpose:

- **Quick Text Creation**: Offers a quick way to create a new text object within the scene with predefined settings.
- **Prefab Usage**: Demonstrates the use of prefabs for creating standardized objects in the scene.
- **Automation of Scene Modification**: Automatically marks the scene as modified to ensure that changes are saved.

### Summary:

The `SceneCreateTextObject` script streamlines the process of creating a new text object within a Unity scene. It leverages a prefab for consistency, allows for parent-child relationships, and ensures that scene modifications are appropriately marked. This can be useful for scenarios where developers frequently need to add standardized text objects to the scene during development.

## CanvasConverter and TMP\_CanvasConverter

This script, `CanvasConverter`, is designed to convert specific UI components from a "Legacy" system to an "LT Legacy" system within Unity's Editor. Let's break down its structure and functionality:

### Variables:

- **selectedObject**: Represents the `GameObject` currently selected in the Unity Editor hierarchy.

### Methods:

- **ConverterUiToLT()**: This method is called when the user selects the menu item "GameObject/Language/Converter/UI/Legacy to LT Legacy" in the Unity Editor.

### Functionality:

#### - Component Checking and Conversion:

- Checks the currently selected `GameObject` for specific UI components: Text, Dropdown, Button, Toggle, and InputField.
- Converts these components from the legacy system to the "LT Legacy" system by attaching relevant "LT Legacy" components and configurations.

### Usage:

- **Menu Item**: The method `ConverterUiToLT()` is associated with the menu item "GameObject/Language/Converter/UI/Legacy to LT Legacy". When this menu item is selected, the script attempts to convert the selected UI components to the new system.

### Purpose:

- **Legacy System Conversion**: Automates the process of converting specific UI components from a legacy system to a new "LT Legacy" system.
- **Prefab Attachment**: Adds and configures new components to existing `GameObjects` based on the presence of specific UI components.

### Summary:

#### - Text Component:

- Attaches a `LanguageText` component to Text objects, transferring the Text's content and marking the scene as modified.

#### - Dropdown Component:

- Adds `LanguageDropdown` and `AdjustSizeToDropdown` components, copying options and adjusting configurations accordingly.

#### - Button Component:

- Attaches `LanguageText` and `AdjustRect` components to the Text and Button objects, respectively.

#### - Toggle Component:

- Adds `LanguageText`, `AdjustRect`, and `AdjustRect` on `targetGraphic` components to the Text, Toggle, and targetGraphic objects, respectively.

#### - InputField Component:

- Attaches `LanguageTextInputField`, `LanguageText`, and `AdjustRect` components to the Text, Placeholder, and InputField objects, respectively.

### Details:

- For each UI component type, it ensures that duplicate conversions or missing components are appropriately handled, providing error messages when necessary.
- Marks the scene as modified after each successful conversion to indicate changes.

### Final Remarks:

The `CanvasConverter` script streamlines the process of updating specific UI components from a legacy system to a new "LT Legacy" system in Unity's Editor. It offers automated component attachment and configuration while ensuring scene modification tracking for accurate saving of changes.

## SceneConverter and TMP\_SceneConverter

This script, **SceneConverter**, facilitates the conversion of TextMesh components from a "Legacy" system to an "LT Legacy" system within Unity's Editor. Let's break down its structure and functionality:

### Variables:

- **selectedObject**: Represents the GameObject currently selected in the Unity Editor hierarchy.

### Methods:

- **ConverterUiToLT()**: This method is invoked when the user selects the menu item "GameObject/Language/Converter/Scene/Legacy to LT Legacy" in the Unity Editor.

### Functionality:

#### - Component Checking and Conversion:

- Checks if the currently selected GameObject contains a TextMesh component.
- If a TextMesh component is found, it attaches and configures the **LanguageTextMesh** component.
- The **LanguageTextMesh** component is linked to the existing TextMesh component for the conversion.

### Usage:

- **Menu Item**: The method **ConverterUiToLT()** is associated with the menu item "GameObject/Language/Converter/Scene/Legacy to LT Legacy". Triggering this menu item attempts to convert the selected TextMesh component to the new system.

### Purpose:

- **TextMesh Conversion**: Facilitates the migration of TextMesh components from the legacy system to a new "LT Legacy" system.
- **Error Handling**: Provides error messages if no object is selected or if the selected object does not contain a TextMesh component.

### Summary:

- Checks for the presence of a TextMesh component in the selected GameObject.
- If a TextMesh component exists, attaches a **LanguageTextMesh** component to it, linking the TextMesh content for the conversion.
- Logs success and error messages accordingly, marking the scene as modified after a successful conversion.

### Details:

- Verifies that the selected object contains a TextMesh component before attempting to convert it to the new system.
- If the selected object already contains the **LanguageTextMesh** component, it displays an error message to prevent duplicate attachments.
- Marks the scene as modified after each successful conversion.

### Final Remarks:

The **SceneConverter** script streamlines the process of converting TextMesh components from a legacy system to a new "LT Legacy" system in Unity's Editor. It ensures accurate conversion by checking the presence of the TextMesh component and appropriately handles error scenarios, indicating scene modifications for saving changes accurately.

## LanguageSettingsWindow

This script, `LanguageSettingsWindow``, manages an editor window in Unity for configuring language-related settings. Let's break down its structure, variables, methods, and usage:

### Variables:

- **scrollPosition**: Tracks the scroll position within the window.
- **Path Variables for JSON Files and Folders**: Defines paths and names for JSON files and folders used in the project.
- **firstTime**: Flag indicating if it's the first time the window is displayed.
- **fontListObject** and **TMP\_fontListObject**: Objects containing font lists used in the project.

### Methods:

- **ShowWindow()**: Displays the window and sets its custom icon.
- **OnGUI()**: Handles the window's GUI layout and interactions.
- **DeleteDataFile()**: Deletes specific language data files.
- **ChangeLabelTextInAllObjects()**: Modifies variables in scripts attached to objects in the scene based on window settings.
- **OnDestroy()**: Called when the window is closed, saves information.
- **SaveDataToFile()** and **LoadDataFromFile()**: Save and load settings from a file.
- **CreateOrReplaceScriptableObject()**: Creates or replaces a ScriptableObject for language settings.

### Functionality:

- **Window Display**:
  - Presents a window for configuring language settings in Unity's Editor.
  - Displays fields for paths, settings, font objects, and buttons for actions.
- **Data Handling**:
  - Loads saved data when the window is first displayed.
  - Saves settings to a file when the window is closed or specific actions are performed.
  - Modifies specific variables in scripts attached to objects in the scene based on user-configured settings.
- **Action Buttons**:
  - Offers buttons to reset settings, apply settings to the current scene, and create language data.

### Usage:

- **Window Creation**: Use the menu item "Window/Language/Language Settings" to display this window in Unity's Editor.
- **Settings Configuration**: Users can configure paths, settings, and assign font objects for language-related functionalities through this window.
- **Actions**: Users can reset settings, apply settings to the scene, and create language data by interacting with the buttons provided in the window.

### Summary:

- Manages an editor window for configuring language settings in Unity.
- Allows users to define paths, settings, and font lists for language functionalities.
- Interacts with scripts attached to objects in the scene to modify variables based on configured settings.

### Additional Details:

- The script handles file operations, modifying script variables, and creating/replacing ScriptableObjects for language settings.

### Final Remarks:

The `LanguageSettingsWindow`` script serves as a versatile tool within Unity's Editor, offering a user-friendly interface for configuring language-related settings, managing files, and applying changes across the scene based on user-defined preferences.



## LanguageSaveEditorWindow

This script is an editor window in Unity for managing language-related data, potentially for localization purposes. Let's break down the components:

### Variables:

- **savePath**: Defines the folder path where language-related data will be saved.
- **fileName**: Indicates the name of the text file.
- **languageName**: Represents the name of the language being managed.
- **computerLanguage**: Indicates the language set on the user's computer.

### Lists:

- **groupsID**: Contains groups of language-related data.
- **textID**: Stores individual text-related data.
- **commentsID**: Holds comments associated with the language data.

### UI Control Variables:

- **scrollPosition**: Manages the scroll position of the window.
- **showDrawing**: Controls the visibility of the design section.
- **firstTime**: Flags if it's the first time the method runs.

### Serialized Object and Helpers:

- **serializedObject**: Manages serialized data for GUI usage.
- **systemLanguage**: Stores the system-recommended language.
- **availableLanguages**: Stores all available computer languages.
- **selectedLanguageIndex**: Tracks the index of the selected language.
- **duplicateIDs**: Contains a dictionary of duplicate IDs.

### Methods and Functionality:

- **OnEnable()**: Initializes the serialized object.
- **OnDestroy()**: Saves information and closes related windows.
- **OnGUI()**: Renders the window interface with various sections.
- **Tools()**: Renders configuration tools for path, file name, language name, etc.
- **Notice()**: Displays a warning if there are duplicate IDs.
- **Button()**: Renders action buttons at the window bottom.
- **SortByID()**: Sorts the data lists by ID.
- **ListInspector()**: Renders watch sections for data lists.
- **Design()**: Renders a design section with colored boxes displaying data.
- **RenderTextBox()**: Renders a box with specific text-related information.
- **OpenFolder()**: Opens the destination folder on the file system.
- **SaveTXTFile()**: Saves data to a text file based on the defined structure.
- **SaveDataJson()** & **LoadDataJson()**: Save and load window settings to/from a JSON file.

### Class Definitions:

- **LanguageSaveGroups**, **LanguageSaveID**, **LanguageSaveComments**: Serializable classes defining the structure of group, text, and comment data.
- **LanguageData**: Defines the overall structure for saving/loading data settings.

The script integrates with Unity's editor to create a custom window for managing language-related content. It offers tools to configure, visualize, save, and load language data in various formats.

## LanguageSaveEditorSettingsWindow

This script is be an editor window in Unity that manages and manipulates various data structures for a language localization system. Here's a breakdown of its key components:

### Variables:

- **instance**: Reference to the editor window instance.
- **scrollPosition**: Manages the scroll position for the GUI.
- **text**: Stores the currently selected item's text.
- **textID, IDs, \_text**: Variables for managing group data like text ID, IDs, and text content.
- **comments, textComments**: Variables for managing comment data like comment ID and text content.
- **groups, \_comments**: Flags for controlling which sections are displayed.
- **selectGroups, selectComments, selectText**: Flags for selecting the type of component (Group, Comment, Text).
- **ID**: Used for deleting selected items.

### Methods:

- **OnGUI()**: Handles the GUI rendering and interaction.
- **AddObject()**: Deals with adding new groups or comments.
- **Design()**: Manages the design and editing of group or comment data.
- **DeleteComponent()**: Handles the deletion of selected components.
- Methods like **AddGroups()**, **GroupsApply()**, **AddComments()**, **CommentsApply()**, **SelectGroups()**, **SelectComments()**, **SelectText()**, **Delete()**: These methods handle specific functionalities related to adding, applying, selecting, and deleting groups, comments, and text.

### Usage:

- **Add new Object**: This section has buttons for adding new groups or comments.
- **Design**: This part handles the editing of group or comment data.
- **Delete Component**: Manages the deletion of selected items.
- **Selecting Components**: Allows selection between Text, Group, or Comment components.
- **Confirmation Dialogs**: Confirm actions before deleting or clearing data.

### Workflow:

1. **Adding Objects**: Use buttons to add groups or comments.
2. **Design Section**: Edit group or comment data.
3. **Delete Component**: Choose and delete selected items.
4. **Confirmation**: Confirm deletion actions.
5. **Applying Changes**: Apply changes to groups or comments.
6. **Saving Data**: Update and save data.

This script primarily provides a GUI interface within Unity's editor for managing and manipulating language localization data, allowing the addition, editing, and deletion of groups, comments, and text components.

## FontAssetBundleCreator

This script in Unity is a static class **FontAssetBundleCreator** that creates asset bundles from fonts within the Unity Editor. Here's a breakdown of its components:

### Variables:

- **fontFolderPath**: Base directory for AssetBundles where fonts are stored.
- **fontPaths, TMP\_fontPaths**: Arrays holding paths to regular fonts and TMP\_FontAssets, found within the **fontFolderPath**.

### Methods:

- **CreateFontAssetBundles()**:
  - Searches for regular fonts and TMP\_FontAssets within the specified directory.
  - Processes regular fonts and TMP\_FontAssets separately.
  - For each font found:
    - Determines the file extension to validate it as a font file.
    - Creates an **AssetBundleBuild** configuration.
    - Builds an asset bundle for each font using **BuildPipeline.BuildAssetBundles**.
    - Logs a success message.
- **IsFontFile(string extension)**:
  - Checks if a given file extension matches common font extensions (**.ttf**, **.otf**, **.ttc**).

### Usage:

1. **Menu Option**: This script is invoked through the Unity Editor's menu with the option "Assets/Create Font Asset Bundles (Language-Unity-Package)".
2. **Asset Bundle Creation**: Searches for regular fonts and TMP\_FontAssets within the specified directory and creates asset bundles for each.
3. **Logging**: Provides success messages via **Debug.Log**.

### Workflow:

1. **Menu Trigger**: The user triggers the menu option in the Unity Editor.
2. **Font Detection**: Finds regular fonts and TMP\_FontAssets in the specified directory.
3. **Asset Bundle Creation**: Processes fonts by creating asset bundles for each font file.
4. **Logging Success**: Logs success messages for each asset bundle created.
5. **Asset Database Update**: Refreshes the Asset Database to reflect changes made.

### Additional Details:

- **AssetBundleBuild**: Configures asset bundle settings like the name and assets included in the bundle.
- **BuildPipeline.BuildAssetBundles**: Unity API method to build asset bundles.
- **AssetDatabase.Refresh()**: Unity API method to refresh the Asset Database to reflect changes made during script execution.

Overall, this script automates the process of creating asset bundles from fonts in Unity, allowing for easier management and distribution of fonts as asset bundles within a project.