



TSV Tools

Documentation

Creator: Lucas Gomes Cecchini

Online Name: AGAMENOM

Overview

TSV Tools is a tool for creating and modifying **.tsv** files for **Unity**.

With it, you can create a text-based data table in the **editor**, and it also allows you to read and write or edit the file in both **Editor** and **Build** mode.

Script Explanations

Tab Table Utility

Overall Purpose

This script provides a set of tools to work with tables where values are separated by tabs, commonly called **TSV tables**. It allows you to:

- Load and save TSV files.
- Read or modify individual cells.
- Add or remove rows and columns.
- Handle special text tokens for tab and newline characters.

The tables are organized as **vertical tables**, where each row contains an array of cell values.

Core Components

1. Enums

- **LineDirection**
 - Represents the direction in which a line operation occurs.
 - Two possible values: Vertical (row operations) and Horizontal (column operations).
 - Used when adding or removing rows or columns.

2. Text Conversion

- **ConvertText(text)**
 - Converts special tokens in text to actual characters and vice versa.
 - Handles:
 - %tab% → tab character
 - %newline% → newline character
 - And the reverse conversion.
 - Usage: Ensures that special characters in table cells can be safely stored or displayed without breaking the tab-separated format.

3. File Operations

- **LoadTableFile(filePath, table)**
 - Loads a TSV file from the given path into a table.
 - Automatically detects UTF-8 encoding, including files with or without a BOM marker.
 - For each line:
 - Splits it by tab characters.
 - Converts the text using the text conversion function.
 - Stores the result in a row object.
 - Usage: Populate a table with external TSV data for reading or manipulation.
- **SaveTableFile(filePath, table)**
 - Saves a table to a TSV file.
 - Ensures all rows have the same number of columns by filling missing cells with empty strings.
 - Converts special characters back to tokens for safe storage.
 - Uses UTF-8 without BOM to maximize compatibility with other tools.
 - Usage: Save the current table state to a file that can be read by other software or reused later.

4. Table Access

- **GetText(table, verticalIndex, horizontalIndex)**
 - Returns the content of a specific cell in the table.
 - Checks that indices are valid; logs errors if they are not.
 - Usage: Retrieve a cell's value for display, calculation, or modification.
- **SetText(table, verticalIndex, horizontalIndex, text)**
 - Updates a specific cell in the table with new text.
 - Performs index validation and logs errors if the indices are invalid.
 - Usage: Change the content of a specific cell in the table.

5. Table Modification

- **AddLine(table, position, direction)**
 - Adds a new line at the specified position.
 - Vertical adds a new row.
 - Initializes with the same number of columns as existing rows.
 - Horizontal adds a new column in all rows.
 - Fills it with empty strings.
 - Usage: Expand the table with additional rows or columns.
- **RemoveLine(table, position, direction)**
 - Removes a line from the table.
 - Vertical removes a row.
 - Horizontal removes a column from all rows.
 - Performs validation to avoid invalid operations.
 - Usage: Shrink the table by removing unwanted rows or columns.

6. Data Structure

- **VerticalTable**
 - Represents a single row of the table.
 - Contains an array of strings called horizontalTable.
 - Each string is a cell value.
 - Usage: Holds the table's actual data in memory.

Variables Overview

- table – the main structure representing the TSV table as an array of rows.
- horizontalTable – the array of strings representing the cells in a single row.
- columnVertical – row index in the table.
- columnHorizontal – column index in the table.
- lineDirection – determines whether an operation affects rows (vertical) or columns (horizontal).
- text – the content of a cell to read or write.
- filePath – path to a TSV file for reading or writing.

Usage Examples

1. **Loading a Table**
 - Read a TSV file into memory for manipulation.
 - All rows and cells are automatically processed and stored as VerticalTable objects.
2. **Reading a Cell**
 - Access a specific value using row and column indices.
3. **Editing a Cell**
 - Change a specific value in memory without immediately affecting the file.
4. **Adding/Removing Rows or Columns**
 - Dynamically expand or shrink the table in memory.
5. **Saving a Table**
 - Write the table back to disk in TSV format, ensuring proper encoding and uniform row length.

TSV Editor Window

Here's a detailed explanation of how the editor script works, describing variables, methods, and usage, **without using C# terminology**:

Overall Purpose

This script creates a custom editor window inside Unity specifically for working with **TSV (Tab-Separated Values) tables**. It allows users to:

- Load existing TSV files into a table.
- Save changes back to a TSV file.
- Edit cell content directly in the editor.
- Add or remove rows and columns.
- Resize rows and columns with interactive handles.
- Scroll through large tables easily.

It acts as a **visual and interactive interface** for managing tabular data inside the Unity editor.

Core Components

1. Public Variables

These hold the main table data and its state:

- **tableData**: The main structure holding the table rows. Each row contains multiple cells.
- **filePath**: Path of the TSV file currently loaded.
- **targetIndex**: Index of the selected row or column for operations (like removal).
- **removalDirection**: Indicates whether the selected line for removal is a row (vertical) or a column (horizontal).
- **rowSizes / columnSizes**: Arrays that store the heights of rows and widths of columns, used for rendering.

2. State Variables

These track the editor's current interactive state:

- **isResizingRow / isResizingColumn**: Track whether a row or column is currently being resized.
- **initialResizePosition**: Stores the mouse position when resizing begins.
- **originalDimension**: Original size of the row/column before resizing.
- **resizeTargetIndex**: The row/column currently being resized.
- **mainScrollPosition / tableScrollPosition**: Positions for scrolling the editor window and table content.
- **isDirty**: True if the table has unsaved changes.
- **previousScrollPosition**: Tracks previous scroll state to manage focus during scrolling.
- **lastInteractionTime**: Records last click time for detecting double-clicks.
- **isValidIndex**: True if the selected row/column index is within the table bounds.

3. Constants and UI Settings

These define visual and interaction rules:

- **Minimum sizes**: minColumnWidth and minRowHeight set limits for resizing.
- **Double-click threshold**: Maximum time to detect a double-click.
- **Colors**: Define appearance of table elements:
 - Default lines, highlight during resize, table background, headers, and selected row/column.
- **GUI styles**: Styles for text inside cells and labels.
- **Textures**: Background textures for table cells.

Key Methods and Their Usage

Menu and Window Management

- **OpenTableEditorWindow()**: Opens the editor window in Unity. Searches for a custom icon and sets it in the window title.

Unity Callbacks

- **OnEnable()**: Subscribes to Undo/Redo events so changes are reflected in the window.
- **OnDestroy()**: Saves temporary table data and unsubscribes from Undo/Redo events.
- **OnUndoRedo()**: Repaints the editor after an Undo or Redo operation.

GUI Rendering

- **OnGUI()**: Main method called every frame to draw the editor interface. Handles:
 - Scroll views.
 - File loading and saving.
 - Table creation.
 - Row and column management.
 - Rendering table content and headers.
- **InitializeStyles()**: Sets up visual styles for cells and headers, including textures for backgrounds.
- **RenderOpenFileButton()**: Opens a file dialog to select a TSV file. Loads table data if a valid file is selected.
- **RenderSaveButton()**: Displays a save button. Highlights it if there are unsaved changes. Saves the table when clicked.
- **RenderRemoveRowColumnControls()**: Allows removing a selected row or column. Validates index and direction.
- **RemoveSelectedLine()**: Removes the specified row or column from the table. Updates row/column size arrays and marks the table as modified.

- **RenderCreateTableButton()**: Creates a default table if no table is loaded. Initializes two rows and two columns with default sizes.

Table Utility Methods

- **CreateTexture(width, height, color)**: Creates a single-color texture used for GUI backgrounds.
- **LoadTableFromFile(path)**: Loads TSV data into the table. Initializes row heights and column widths.
- **SaveTableToFile(path)**: Saves the table to a TSV file and marks the table as not dirty.

Table Rendering

- **RenderTable()**: Draws the entire table, including:
 - Column headers and row labels.
 - Scrollable table content.
 - Editable text fields for each cell.
 - Resizing handles for rows and columns.
 - Double-click detection to add new rows or columns.
 - Highlights selected row/column for removal.
- **HandleResizeInteraction(event)**: Handles resizing of rows and columns by dragging handles. Ensures sizes do not go below the minimum.

Variables During Rendering

- **xOffset / yOffset**: Track positions when rendering cells and headers.
- **totalWidth / totalHeight**: Total dimensions of the table for scrollable area.
- **visibleRect**: Defines currently visible part of the table for optimization.
- **cellRect / rowLabelRect / colResizeHandle / rowResizeHandle**: Rectangles representing positions and sizes of cells, labels, and resizing handles.

Usage Workflow

1. Open the editor window from Unity's menu.
2. Use the **Open File** button to load an existing TSV file.
3. The table is displayed with scrollable content, editable cells, and headers.
4. Users can:
 - Edit individual cells.
 - Resize rows and columns by dragging handles.
 - Double-click headers or edges to add new rows or columns.
 - Select a row or column and remove it using controls.
5. Changes are tracked using the **isDirty** flag.
6. Use the **Save Table** button to save modifications back to a TSV file.

This editor provides a **fully interactive TSV table management system** directly inside Unity, making it easy to visualize, edit, and organize tabular data without external software.

Test Table

Overall Purpose

This script is designed to test and interact with **TSV (Tab-Separated Values) tables** inside Unity. It allows a user to:

- Load a TSV file into a table structure.
- Save the table back to a TSV file.
- Read and edit specific cells.
- Add or remove rows and columns.
- Use buttons in the Unity Inspector to perform all operations easily.

It works as a **bridge between the table data and Unity's Inspector**, making table management interactive without opening external programs.

Variables and Their Roles

1. Table Data

- **table**: Holds the loaded table, organized by rows, each containing multiple cells.
- **filePath**: Stores the path to the TSV file currently loaded.

2. Cell Operations

- **text**: The string used for reading from or writing to a specific cell.
- **columnVertical**: Row index for the selected cell.
- **columnHorizontal**: Column index for the selected cell.

3. Add/Remove Operations

- **direction**: Determines if the operation affects a row (vertical) or a column (horizontal).
- **addColumn**: Position index where a row or column will be added or removed.

4. Properties

For each main variable, there is a corresponding getter/setter that allows other parts of the program to access or modify the values safely:

- **Table, FilePath, Text, ColumnVertical, ColumnHorizontal, Direction, AddColumn.**

Key Methods and Usage

File Operations

- **GetFile()**: Opens a file dialog to select a TSV file. After selecting a file, the table is loaded into memory and its content becomes editable. Logs success or warning if no file is selected.
- **SaveFile()**: Opens a save file dialog to store the table into a TSV file. Checks first if a table exists. Logs the save path or cancellation warning.

Cell Operations

- **GetText()**: Reads the content of a specific cell based on row and column indices and stores it in the text variable. Logs the retrieved content to the console.
- **SetText()**: Writes the content from the text variable into the selected cell at the specified row and column. Logs the update to the console.

Table Modification

- **AddColumnInTable()**: Inserts a new row or column at the specified position depending on the direction.
- **RemoveColumnInTable()**: Deletes a row or column at the specified position depending on the direction.

Validation Helper

- **Checking()**: Ensures a table is loaded and not empty before performing operations. Logs an error if the table is invalid. Returns true if the table is not ready.

Inspector Buttons

A custom interface is drawn in Unity's Inspector to make the operations easy:

1. **File operations row:**
 - **Get File**: Load a TSV file.
 - **Save File**: Save the current table.
2. **Cell operations row:**
 - **Get Text**: Read content from a specific cell.
 - **Set Text**: Write content into a specific cell.
3. **Add/Remove operations row:**
 - **Add Column**: Insert a new row or column at the specified index.
 - **Remove Column**: Remove a row or column at the specified index.

Below the buttons, the default Inspector shows all the serialized variables, allowing the user to see or edit the row/column indices, text, and direction.

Usage Workflow

1. Attach this script to an object in a Unity scene.
2. Open the Inspector for that object.
3. Click **Get File** to load a TSV table.
4. Optionally, use **Get Text** to read a cell or **Set Text** to modify a cell.
5. Add or remove rows/columns using the buttons and select the correct direction and index.
6. Click **Save File** to save your changes to disk.
7. The console logs provide feedback for all actions.

Summary

This test script provides a **simple and interactive way to manage TSV tables directly from Unity**, using buttons in the Inspector for file operations, cell editing, and structural changes to the table. It acts as a lightweight editor for testing the functionality of the table management system.