PlayerPrefsRuntime Tool

`version 2.0`

Overview

PlayerPrefsRuntime Tool is a Unity plugin that allows you to retrieve all `PlayerPrefs` at runtime across multiple platforms, including Android, iOS, Windows, and macOS. This tool is invaluable for debugging, analytics, and ensuring the integrity of player preferences within your Unity projects.

Features

- **Cross-Platform Support:** Compatible with `Android`, `iOS`, `Windows`, and `macOS`.
- **Runtime Access:** Retrieve all `PlayerPrefs` as a dictionary directly from the device for easy manipulation.
- **Logging:** Display all `PlayerPrefs` data in the Unity Console for debugging purposes.
- **Interactive UI Viewer:** Visual interface with search, sorting, and detailed entry inspection.
- **Extensible Architecture:** Easily extendable to support additional platforms or functionalities.

UI Viewer Features

The PlayerPrefsRuntime Tool includes a comprehensive visual UI viewer that provides:

Real-time Search & Filtering

- Live search across key names, types, and values
- Case-insensitive filtering with instant results
- Clear button to reset search
- Shows filtered count vs total count in header

Sorting Capabilities

- Toggle between "Sort: Name" and "Sort: Type" modes
- Secondary sorting (type within name, or name within type)
- Alphabetical ordering using ordinal comparison

Visual Design & Theming

- Dark theme with accent colors (blues, cyans)
- Alternating even/odd row colors for readability
- Color-coded type badges (Int, String, Float)
- Interactive rows with hover effects
- Safe area support for mobile devices

Interactive Features

- Click on any entry to view detailed information in a dialog
- Close button to hide the viewer and clean up resources
- Sort button to toggle between sorting modes
- Search field with placeholder text

Performance Optimized

- Smooth scrolling with inertia and deceleration
- Clamped movement type (no overscroll)
- Optimized content sizing with proper masking
- Handles large datasets (100+ entries) efficiently

Notes & Limitations

- Windows builds decode the PlayerPrefs registry entries Unity writes. Standard `int`, `float`, and `string` keys are returned; values stored in any other registry format are skipped and reported in a single aggregated warning so logs stay readable.
- iOS/macOS rely on a native JSON export. The helper normalizes numeric types back to Unity-friendly values (e.g. large `long` stay `long`, doubles collapse to `float`), so downstream code sees the usual `PlayerPrefs` types.
- Define `PLAYER_PREFS_RUNTIME_TOOL` only for targets where the native bridge or registry access exists; editor playback will safely fall back to empty results.
- The UI Viewer is automatically disabled in editor playback mode and will show an empty state with a message indicating that PlayerPrefs are not accessible in the editor.

Platform Support Details

Platform	Implementation	UI Viewer	Method
Android	SharedPreferences via AndroidJavaObject	✅	Runtime
iOS	Native P/Invoke to iOS key-value store	✅	Runtime
macOS	Native P/Invoke to macOS preferences	✅	Runtime
Windows	Registry access with binary parsing	✅	Runtime
Editor (Win)	Registry access	✅	Editor only
Editor (Mac)	plist file parsing	✅	Editor only

Platform-Specific Features:

Android:

- Accesses `SharedPreferences` using package name
- Handles URI-encoded keys and values
- Supports standard int, float, string types

iOS/macOS:

- Native JSON export via P/Invoke
- Requires native plugins in `Assets/Plugins/`
- Handles binary data and complex types

Windows:

- Registry-based access at `Software\{company}\{product}`
- Custom binary parsing for Unity's PlayerPrefs format
- Supports int, float, string, and binary data

Data Type Handling:

- Automatic normalization across platforms
- Type consistency: Ensures long→int, double→float conversions
- Binary data support: Base64 encoding/decoding where needed
- Null safety: Graceful handling of missing or corrupted data

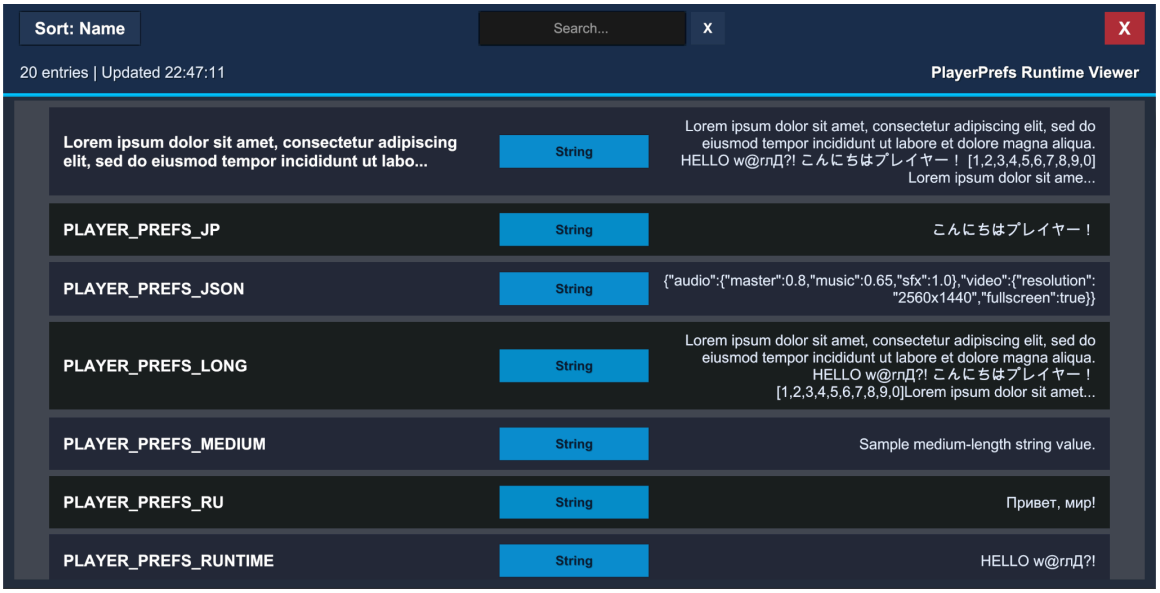
---

1. Enable the Tool

To enable the PlayerPrefsRuntime Tool, define the `PLAYER_PREFS_RUNTIME_TOOL` scripting symbol:

Go to `Edit > Project Settings > Player`. Under the Other Settings tab, find Scripting Define Symbols. Add PLAYER_PREFS_RUNTIME_TOOL to the list, separated by a semicolon if other symbols are present.

2. UI Viewer Setup (Optional)



The UI Viewer is automatically configured and requires no additional setup. However, you can customize the appearance by modifying the `PlayerPrefsRuntimeViewConstants.cs` file:

Color Customization:

- Panel colors: `PanelColor`, `HeaderColor`, `AccentColor`
- Row colors: `RowColorEven`, `RowColorOdd`
- Text colors: `ValueTextColor`, `BadgeLabelColor`
- Control colors: `ControlNormalColor`, `CloseButtonNormalColor`

Layout Customization:

- Row dimensions: `RowMinHeight`, `RowSpacing`, `RowPaddingHorizontal`
- Font sizes: `NameFontSize`, `ValueFontSize`, `BadgeFontSize`
- Text limits: `MaxNameTextLength`, `MaxValueTextLength`

Performance Settings:

- Scroll sensitivity: `ScrollSensitivity`
- Scroll deceleration: `ScrollDecelerationRate`
- Canvas scaling: `CanvasMatchWidthOrHeight`

The UI Viewer automatically creates required components (Event System, Canvas) if they don't exist.

Usage

Basic API Usage

```
using System.Collections.Generic;
using UnityEngine;
using DmytroUdovychenko.PlayerPrefsRuntimeTool;

public class PlayerPrefsRuntimeExample : MonoBehaviour
{
    private void Start()
    {
#if PLAYER_PREFS_RUNTIME_TOOL
        // Add test data for demonstration
        PlayerPrefsRuntime.AddTestPlayerPrefs();

        // Log all PlayerPrefs to console
        PlayerPrefsRuntime.LogAllPlayerPrefs();

        // Retrieve all PlayerPrefs as Dictionary <key, value>
        Dictionary<string, object> allPrefs = PlayerPrefsRuntime.GetAllPlayerPrefs();

        // Show the interactive UI viewer
        PlayerPrefsRuntime.ShowAllPlayerPrefs();
#endif
    }
}
```

Demo Scene

The included demo scene (`Assets/DmytroUdovychenko/PlayerPrefsRuntimeTool/Demo/DemoScene.unity`) showcases:

Test Data Generation

- String types: Short, medium, long, very long (100+ lines), Unicode (Russian, Japanese)
- Numeric types: Integers (positive, negative, zero), Floats (small, large, decimal)
- Special cases: JSON data, empty values, keys with special characters

Interactive Demo

- Press **"P" key** to toggle viewer visibility
- Automatically shows viewer after 0.5s delay on start
- Demonstrates complete workflow: add data → retrieve → display

Key Demo Features

- Handling of Unicode and special characters
- Large dataset performance (100+ entries)
- Cross-platform data consistency
- Search and sorting with diverse data types

To use the demo:

1. Open the demo scene
2. Press Play in the Unity Editor
3. Press "P" to toggle the UI viewer
4. Use search and sort features to explore the data

License

Contact

For any questions, suggestions, or feedback, please contact:

- **Linkid-in:** https://www.linkedin.com/in/dmytro-udovychenko/