PlayerPrefsRuntime Tool

`version 3.0`

Overview

PlayerPrefsRuntime Tool is a Unity plugin that allows you to retrieve and edit all `PlayerPrefs` at runtime across multiple platforms, including Android, iOS, Windows, and macOS (with Windows/macOS also supported in the Unity Editor). This tool is invaluable for debugging, analytics, testing, and managing player preferences within your Unity projects.

Dependencies

PlayerPrefsRuntime Tool is using a json parser: "com.unity.nuget.newtonsoft-json": "3.2.1".

Features

- **Cross-Platform Support:** Compatible with `Android`, `iOS`, `Windows`, and `macOS` (plus Windows/macOS Editor).
- **Runtime Access:** Retrieve all `PlayerPrefs` as a dictionary directly from the device/build (or editor storage on Windows/macOS).
- **Live Editing:** Edit, save, and delete `PlayerPrefs` entries directly from the runtime UI.
- **Logging:** Display all `PlayerPrefs` data in the Unity Console for debugging purposes.
- **Interactive UI Viewer:** Visual interface with search, sorting, detailed entry inspection, and edit mode.
- **Type Validation:** Automatic validation for Int, Float, and String values with error feedback.
- **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)
- Safe area support for mobile devices

Interactive Features

- Click on any entry to view detailed information in a dialog
- Toggle edit mode with the "Edit" button

- Modify Int, Float, and String values directly
- Save changes with the "Save" button
- Delete entries with the "Delete" button
- Real-time validation with error messages
- Automatic UI refresh after saving or deleting
- Close button to hide the viewer and clean up resources
- Sort button to toggle between sorting modes
- Search field with placeholder text

Notes & Limitations

- The tool is compiled conditionally behind the `PLAYER_PREFS_RUNTIME_TOOL` define constraint (see `PlayerPrefsRuntime.asmdef`). If the define isn't set, the API won't exist in your project.
- Windows reads `PlayerPrefs` from the registry `(HKCU\Software\{company}\{product}` in player, and `HKCU\Software\Unity\UnityEditor\{company}\{product}` in editor) and strips Unity's hashed suffix from keys (e.g. `_h123`). Returned values are best-effort decoded and may include `int, long, float, string`, or `byte[].`
- Android reads SharedPreferences from `${Application.identifier}.v2.playerprefs` and supports `int, float`, and `string` (other types are stringified).
- iOS exports only `NSString/NSNumber` values from `NSUserDefaults` (other types are skipped). macOS runtime reads `~/Library/Preferences/.plist` and serializes it to JSON; non-JSON plist types can cause an empty result. macOS Editor parsing supports XML and binary plists (`via plutil`).

Platform Support Details

Platform	Implementation	UI Viewer	Method
Android	`PlayerPrefsRuntimeFetcherAndroid` - SharedPreferences via AndroidJavaObject	✅	Runtime
iOS	`PlayerPrefsRuntimeFetcherIOS` - Native P/Invoke to UserDefaults	✅	Runtime
macOS	`PlayerPrefsRuntimeFetcherMacOS` - Native P/Invoke to UserDefaults	✅	Runtime
Windows	`PlayerPrefsRuntimeFetcherWindows` - Registry access with binary parsing	✅	Runtime
Editor (Win)	`PlayerPrefsRuntimeFetcherWindowsEditor` - Registry access	✅	Editor only

Editor (Mac)	`PlayerPrefsRuntimeFetcherMacOSEditor` - plist file parsing (XML/binary)	✅	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 `(PlayerPrefsRuntimePlugin.mm)`
- Native plugins located in `Assets/DmytroUdovychenko/PlayerPrefsRuntimeTool/Plugins/iOS/` and `Plugins/macOS/`
- iOS exports only strings/numbers from UserDefaults; macOS runtime depends on plist values being JSON-serializable

Windows:

- Registry-based access at `Software\{company}\{product}`
- Custom binary parsing for Unity's PlayerPrefs format
- Best-effort decoding: `int, long, float, string`, and sometimes `byte[]`

Data Type Handling:

Automatic normalization across platforms via `PlayerPrefsRuntimeJsonHelper`
Type consistency: Ensures long→int, double→float conversions
Null safety: Graceful handling of missing or corrupted data

What's New in Version 3.0

Edit Mode

- The biggest addition in v3.0 is the ability to edit PlayerPrefs at runtime:
- Toggle Edit Mode: Click any entry to open the detail dialog, then click "Edit" to enable editing
- Type-Safe Editing: Validates input based on the entry type (Int, Float, String)
- Real-Time Validation: Shows error messages for invalid input before saving
- Save Changes: Click "Save" to apply changes directly to PlayerPrefs
- Delete Entries: Remove unwanted entries with the "Delete" button
- Auto Refresh: UI automatically updates after saving or deleting
- This makes it easy to test different values, fix corrupted data, or manage player preferences during development and debugging.

Use Cases

- Testing: Quickly modify values to test different game states
- Debugging: Fix corrupted or incorrect PlayerPrefs data on the fly
- QA: Adjust test parameters without rebuilding the app
- Development: Iterate faster by changing values in real-time

Installation

1. **Download the Plugin**

Option A (recommended): Get from Unity Asset Store

PlayerPrefs Runtime Tool with Viewer

Option B: Import the Unity package `PlayerPrefsRuntimeTool.unitypackage` via `Assets > Import Package > Custom Package.`

Option C: Clone the repository and copy `Assets/DmytroUdovychenko/PlayerPrefsRuntimeTool` into your Unity project:

```
git clone https://github.com/Udovychenko-Dmytro/PlayerPrefs-Runtime-Tool.git
```

2. 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.

3. UI Viewer Setup (Optional)

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

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
	// (Optional) create some data
	PlayerPrefs.SetInt("TEST_INT", 42);
	PlayerPrefs.SetFloat("TEST_FLOAT", 3.14159f);
	PlayerPrefs.SetString("TEST_STRING", "Hello");
	PlayerPrefs.Save();

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

	// Log all PlayerPrefs to console
	Debug.Log($"Found {allPrefs.Count} PlayerPrefs entries");
	PlayerPrefsRuntime.LogAllPlayerPrefs();

	// 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, Ukrainian, Japanese)
- Numeric types: Integers (positive, negative, zero), Floats (small, large, decimal)
- Special cases: JSON data, keys with special characters and Unicode

Interactive Demo

- Press "P" key to toggle the viewer (close via the X button)
- Automatically shows viewer after 0.5s delay on start
- Demonstrates complete workflow: add data → retrieve → display → edit
- Try editing: Click any entry, press "Edit", modify the value, and save

Key Demo Features

- Full Unicode support (Cyrillic, Japanese, special characters)
- Large dataset performance (100+ entries)
- Cross-platform data consistency
- Search and sorting with diverse data types
- Live editing and validation

To use the demo:

1. Open the demo scene

2. Press Play in the Unity Editor
3. Press "P" to show the UI viewer (close via X)
4. Use search and sort features to explore the data

License

API Reference

Method	Description
`PlayerPrefsRuntime.GetAllPlayerPrefs()`	Returns Dictionary<string, object> with all PlayerPrefs
`PlayerPrefsRuntime.LogAllPlayerPrefs()`	Logs all entries to Unity Console
`PlayerPrefsRuntime.ShowAllPlayerPrefs()`	Shows the interactive UI viewer
`PlayerPrefsRuntime.IsVisible`	Returns true if the UI viewer is currently visible

Contact

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

LinkedIn: https://www.linkedin.com/in/dmytro-udovychenko/