PopupFlow - Queue-Based Popup SDK for Unity

Overview

The **PopupFlow SDK** is a lightweight, queue-based popup system for Unity. It provides:

- Centralized popup management (Singleton)
- Queue support (one popup visible at a time)
- Fully customizable title, description, and button texts
- Confirm / Cancel callbacks
- Show / Hide lifecycle callbacks
- Smooth animations using **DOTween**
- Automatic `EventSystem` setup

This SDK is designed to be **safe, reusable, and scalable** across multiple scenes and gameplay systems.

Architecture

```
PopupManager (Singleton)
  │
  ├── PopupData (Data Model)
  │
  └── PopupUI (Visual Controller)
```

Responsibilities

Component	Responsibility
PopupData	Holds popup content & callbacks
PopupUI	Handles UI rendering, buttons, animations
PopupManager	Manages queue, defaults, and public API

Requirements

- Unity UI (Canvas, Button, TMP)
- **TextMeshPro**
- **DOTween** (Animations)

---

1. PopupData

Purpose

`PopupData` is a **pure data container** representing a popup configuration.

Fields

```
public string Title;
public string Description;

public string ConfirmButtonText;
public string CancelButtonText;

public Action OnConfirm;
public Action OnCancel;
public Action OnShow;
public Action OnHide;
```

Lifecycle Callbacks

Callback	Trigger
OnShow	When popup becomes visible
OnConfirm	When confirm button is clicked
OnCancel	When cancel button is clicked
OnHide	After popup is fully hidden

Clone()

Used internally to prevent mutation of default popup data.

---

2. PopupUI

Purpose

Handles **visual presentation only**:

- UI binding
- Button clicks
- Show / Hide animations

> PopupUI never decides *when* to show a popup. That logic lives in `PopupManager`.

Inspector References

- CanvasGroup (fade)
- Popup Root (scale animation)
- Title & Description TMP texts
- Confirm / Cancel buttons

Public API

Show(PopupData data)

- Applies popup data
- Registers button callbacks
- Plays show animation
- Invokes `OnShow`

Hide()

- Plays hide animation
- Invokes `OnHide`
- Notifies `PopupManager` via `OnPopupHidden`

Cancel Button Logic

- Cancel button is **automatically hidden** if:
    - `CancelButtonText` is null or empty

---

3. PopupManager

Purpose

The **central brain** of the popup system.

Features:

- Singleton-based access
- Queue support
- Default values
- Multiple overloads for developer convenience

Singleton Behavior

PopupManager.Instance.Show(...);

- Persisted using `DontDestroyOnLoad`
- Only one instance allowed

Default Popup Data

Set in Inspector or auto-generated at runtime:

Title: "Alert"
Description: "Something happened"
Confirm: "OK"
Cancel: "Cancel"

Defaults are **cloned** for every popup.

Queue System (Important)

Rules:

- Only **one popup** is visible at a time
- New popups are **queued** automatically
- Next popup shows only after current popup is hidden

Example Flow:

Popup A → Popup B → Popup C

No overlap. No UI conflicts.

Core Show Method

```
Show(
    string title,
    string description,
    string confirmButton,
    string cancelButton,
    Action onConfirm,
    Action onCancel,
    Action onShow,
    Action onHide
)
```

All parameters are optional.

Common Usage Examples

Simple Alert

```
PopupManager.Instance.Show("Warning");
```

Title + Description

```
PopupManager.Instance.Show(
    "Network Error",
    "Please check your internet connection"
);
```

Confirm Action

```
PopupManager.Instance.Show(
    "Retry?",
    "Request failed",
    "Retry",
    () => RetryRequest()
);
```

Confirm + Cancel

```
PopupManager.Instance.Show(
    "Exit Game",
    "Are you sure?",
    "Yes",
    "No",
    OnExitConfirmed,
    OnExitCancelled
);
```

Full Lifecycle

```
PopupManager.Instance.Show(
    "Pause",
    "Game Paused",
    "Resume",
    "Quit",
    OnResume,
    OnQuit,
    () => Time.timeScale = 0,
    () => Time.timeScale = 1
);
```

Cancel Button Rules

Condition	Result
cancel text OR onCancel exists	Cancel button visible
both missing	Cancel button hidden

Force Hide

```
PopupManager.Instance.Hide();
```

Immediately hides the active popup.

EventSystem Handling

PopupManager automatically creates an `EventSystem` if missing.

EnsureEventSystemExists();

This prevents broken UI interaction in new or empty scenes.

Best Practices

- Use PopupManager only (never call PopupUI directly)
- Keep popup callbacks lightweight
- Use queue behavior intentionally (don't spam popups)
- Pause gameplay inside `OnShow`, resume in `OnHide`

Common Use Cases

- API retry dialogs
- Network error alerts
- Exit confirmations
- Pause / Resume
- Tutorial hints
- System-level warnings

Summary

✔ Queue-safe popup handling
✔ Clean separation of concerns
✔ Highly reusable & extensible
✔ Production-ready architecture

This Popup SDK is suitable for **games, apps, tools, and WebGL projects**.

Author: Abhishek Sahu