

BLInformatique — RealFPS_MissionSystem – User Guide

Create structured, event-driven missions for RealFPS
Unity 6000.6.0f1



The banner features a dark, atmospheric background with a forest, a cabin, and a watchtower. In the center, a green metal briefcase with a white flag icon is surrounded by floating bullets. To the left, a tablet displays a mission list. To the right, a hand holds a handgun. The text 'RealFPS' is in large white and blue letters, with 'MissionSystem' in a green box below it. Below the title, it says 'CREATE. TRACK. COMPLETE. IMMERSIVE MISSIONS FOR YOUR GAMES.'

Current Mission
Investigate the house

- ☒ Find the key
- ☒ Open the door
- ☐ Explore the house
- ☐ Eliminate the threat
- ☐ Report to base

Features:

- Multiple Objective Types
- Flexible Order
- Mission Journal
- Trigger Zones
- Door & Key Objectives
- Enemy & Kill Objectives
- Save System Integration

Key Features:

- COMPLETE MISSION SYSTEM
- EASY SETUP & INTEGRATION
- MODULAR & EXTENSIBLE
- FULL SOURCE CODE

Table of Contents

Introduction

-  Overview
-  How To Use

Optional Sections (12)

-  Installation & Quick Start
-  Mission Architecture & MissionData
-  Objective Types & Configuration
-  MissionManager & Notifications
-  Debugging Missions
-  Delivery Cargo Objectives
-  Mission User Interfaces
-  Save System Integration
-  General Troubleshooting
-  Frequently Asked Questions
-  Compatibility
-  Conclusion & Support

Components (12)

Components Overview

BLInformatique.RealFPS (11)

- MissionData
- MissionDebugStarter
- MissionDeliveryZone
- MissionJournalUI
- MissionManager
- MissionObjectiveEntryUI
- MissionObjectiveEventListener
- MissionObjectiveNotifier
- MissionPopupUI
- MissionTrackerUI
- MissionTriggerZone

BLInformatique.RealFPS.Missions (1)

- DestructibleMissionNotifier

Overview

Overview

RealFPS Mission System is an optional addon for **RealFPS Core** that lets you build structured, event-driven missions directly in Unity. Missions are created with `MissionData` assets, executed by a `MissionManager`, and updated by reusable notifier components placed in your scenes.

The system supports linear missions as well as objectives that may be completed out of order. It can track pickups, enemy eliminations, trigger zones, locked doors, destructible objects, deliveries, interactive objects, and custom gameplay events. Runtime interfaces can display the current mission, objective progress, notifications, and a complete mission journal.

Important: RealFPS Mission System is an addon. **RealFPS Core is required and must be imported first.** The Save System and Moving Platform System integrations described in this guide are optional and are not included with this package.

Main Features

- ScriptableObject-based mission creation with `MissionData`.
- Multiple missions and multiple objectives per mission.
- Sequential or out-of-order objective completion.
- Counter-based objectives such as eliminating several enemies or activating several targets.

- Ready-to-use notifications for pickups, enemies, trigger zones, doors, destructible objects, deliveries, and interactive events.
- Custom objective completion through public methods and UnityEvents.
- Mission tracker, popup notifications, journal, and general mission UI components.
- Runtime events for mission and objective start, progress, completion, and failure workflows.
- Optional persistence through **RealFPS Save System**.
- Optional mission events from **RealFPS Moving Platform System**.
- Inspector tools, debug options, scene gizmos, and a dedicated Mission System menu.

Requirements

Requirement	Status	Notes
Unity	Required	Unity 6000.6.0f1 or a compatible Unity 6 release.
RealFPS Core	Required	Import and configure RealFPS Core before installing this addon.
TextMeshPro	Required for supplied UI	Used by the tracker, popup, journal, and mission UI components.
RealFPS Save System	Optional	Adds mission and objective persistence between save sessions.
RealFPS Moving Platform System	Optional	Allows platform events to notify mission objectives.

How the System Works

A typical mission uses four layers:

1. **Mission definition** — A `MissionData` asset stores the mission title, description, objectives, identifiers, target counts, and completion rules.
2. **Mission execution** — A `MissionManager` starts missions, validates objective notifications, tracks progress, and advances the mission sequence.
3. **Scene notifications** — Components such as `MissionObjectiveNotifier`, `MissionTriggerZone`, `MissionDeliveryZone`, and `DestructibleMissionNotifier` report gameplay actions to the manager.
4. **Player feedback** — Components such as `MissionTrackerUI`, `MissionPopupUI`, `MissionJournalUI`, and `MissionUI` display the current state to the player.

The most important link across the system is the **Objective ID**. The identifier entered in a scene notifier must exactly match the corresponding objective identifier in `MissionData`. IDs are case-sensitive; use short, unique, stable values such as `find_key`, `open_bunker`, or `destroy_generator`.

Recommended Workflow

1. Import and configure **RealFPS Core**.
2. Import **RealFPS Mission System**.
3. Open **Tools/BLInformatique/Real FPS/Setup Center** and run the Mission System setup or repair operation.
4. Create a `MissionData` asset and define its objectives.
5. Add or create a `MissionManager` in the gameplay scene.
6. Assign the mission assets to the manager in the intended order.
7. Add the appropriate notifier components to scene objects.
8. Copy the exact Objective IDs from `MissionData` into the notifier components.
9. Configure the tracker, popup, journal, or other UI components required by the project.
10. Test the complete mission flow in a clean gameplay scene before adding persistence.

Run the Setup Center again after installing, updating, or removing a RealFPS addon. This refreshes integrations and scripting defines without creating a hard dependency on optional packages.

How to Use



How to Use This Guide

Start with **Installation & Quick Start**, then follow the chapters in the table of contents according to the mission features required by your project.

Each chapter is independent and can be revisited from the HTML navigation or PDF table of contents. Configure and test one mission feature at a time before adding Save System persistence or advanced integrations.



Optional Sections



Installation & Quick Start

1. Import RealFPS Core

RealFPS Mission System is an addon and does not replace the core package. Import and configure **RealFPS Core** before importing this package. Resolve all Console errors before continuing.

2. Import RealFPS Mission System

Import the Mission System package into the same Unity project. Keep the default folder structure so that prefabs, Resources, scripts, documentation, and optional integrations remain discoverable.

The expected addon folder is:

```
Assets/BLInformatique/Real FPS/Addons/RealFPS_MissionSystem
```

3. Refresh the RealFPS configuration

Open **Tools/BLInformatique/Real FPS/Setup Center** and run the Mission System setup or repair operation. Run the Setup Center again whenever you install, update, or remove a RealFPS addon.

After setup, let Unity finish compiling and confirm that the Console contains no red errors before opening a gameplay scene.

4. Verify the Mission System menu

The following menu must be available:

Tools/BLInformatique/Real FPS/Addons/  Mission System

It provides commands for creating the Mission Manager, Mission Trigger Zone, and Mission Delivery Zone, plus shortcuts for adding notifier components to selected GameObjects.

Quick Start — First Mission in Five Minutes

This example creates a minimal mission completed when the player enters a trigger zone. It is intentionally independent from inventory, weapons, enemies, Save System, and other optional addons.

Step 1 — Prepare a test scene

1. Open a scene containing a configured RealFPS Player.
2. Confirm that the Player GameObject uses the **Player** tag.
3. Enter Play Mode once and verify that the player can move normally.
4. Exit Play Mode before creating the mission objects.

Step 2 — Create the mission asset

1. In the Project window, create a folder for the demo mission, for example `Assets/Missions/TestMission`.
2. Right-click inside the folder.
3. Select **Create/BLInformatique/Real FPS/Missions/Mission Data**.
4. Rename the asset `Mission_FirstSteps`.
5. Configure the mission:

Field	Value
Mission Id	<code>mission_first_steps</code>
Mission Title	<code>First Steps</code>
Mission Description	<code>Reach the marked area.</code>
Auto Start	Enabled
Auto Start Next Mission	Enabled
Objective Order Mode	<code>Ordered</code>

6. Set the **Objectives** array size to `1`.
7. Configure objective element `0`:

Field	Value
Objective Id	<code>reach_test_zone</code>
Title	<code>Reach the Test Zone</code>
Description	<code>Walk into the marked area.</code>

Field	Value
Objective Type	ReachZone
Hide Until Active	Disabled
Auto Complete	Enabled
Require Active Objective	Disabled

Mission IDs and Objective IDs are case-sensitive. Avoid spaces and reuse the exact same Objective ID on every connected scene component.

Step 3 — Create the Mission Manager

1. Select **Tools/BLInformatique/Real FPS/Addons/🎯 Mission System/Create/🎯 Mission Manager**.
2. Select the created `MissionManager` GameObject.
3. Set **Missions** array size to `1`.
4. Assign `Mission_FirstSteps` to element `0`.
5. Keep **Starting Mission Index** and **Starting Objective Index** at `0`.
6. Enable **Auto Initialize On Start**.

The manager automatically searches for compatible RealFPS inventory, weapon, and mission UI components when their explicit references are empty. For this ReachZone test, inventory and weapon references are not required.

Step 4 — Create the trigger zone

1. Select **Tools/BLInformatique/Real FPS/Addons/🎯 Mission System/Create/📍 Mission Trigger Zone**.
2. Move and scale the created zone so the player can walk through it.
3. Confirm that its Collider is configured as a trigger.
4. Set **Objective Id** to `reach_test_zone`.
5. Set **Required Tag** to `Player`.
6. Enable **Trigger Only Once**.
7. Optionally enable the gizmo and debug log options while testing.

The zone can locate the active `MissionManager` automatically when its manager reference is empty. An explicit reference is preferable when a scene contains more than one manager.

Step 5 — Test the complete flow

1. Enter Play Mode.
2. Confirm that the `First Steps` mission starts automatically.
3. Walk the player into the Mission Trigger Zone.
4. Confirm that `Reach the Test Zone` completes.
5. Check the Console if debug logs are enabled.
6. Inspect the Mission Manager during Play Mode to confirm that the mission reaches its completed state.

Quick Troubleshooting

If the objective does not complete, verify the following in order:

- The Mission Manager contains the correct `MissionData` asset.
- **Auto Initialize On Start** is enabled.
- The mission and objective indexes are both `0`.
- The zone Objective ID is exactly `reach_test_zone`.
- The player is tagged **Player**.
- The zone Collider has **Is Trigger** enabled.
- At least one object involved in the trigger collision has the required physics setup.
- No Console error is preventing the Mission System from running.

Do not add Save System integration until this basic mission works correctly. First validate the mission flow, then add persistence and test save/load in a second pass.

Mission Architecture & MissionData

RealFPS Mission System separates mission content, runtime state, scene notifications, and player feedback. This keeps missions reusable and prevents scene objects from owning the complete mission logic.

Layer	Main Components	Responsibility
Definition	<code>MissionData</code>	Stores mission information, objective definitions, completion rules, and mission-specific UnityEvents.

Layer	Main Components	Responsibility
Runtime	<code>MissionManager</code>	Starts missions, validates notifications, tracks counters, advances objectives, and exposes global events.
Scene	Notifiers, trigger zones, pickups, enemies, toggles, doors, destructibles, delivery zones	Reports gameplay actions to the Mission Manager.
Presentation	<code>MissionTrackerUI</code> , <code>MissionPopupUI</code> , <code>MissionJournalUI</code> , <code>MissionUI</code>	Displays mission information without controlling progression.
Persistence	RealFPS Save System integration	Optionally captures and restores the Mission Manager and supported notifier states.

A scene object should report what happened; the Mission Manager decides whether that action currently satisfies an objective. UI components only display the resulting state.

Runtime Flow

1. The Mission Manager initializes the selected `MissionData` .
2. The current objective invokes its start events.
3. A gameplay component reports an action or state change.
4. The Mission Manager compares the notification with the active objective or an eligible future Flexible objective.
5. Matching progress is recorded and the objective-reached events are invoked.
6. When **Auto Complete** is enabled, the objective completes immediately.
7. The manager advances to the next objective or mission and refreshes the UI.

Creating MissionData

Create a mission asset through:

Create/BLInformatique/Real FPS/Missions/Mission Data

Each asset represents one mission. A Mission Manager can contain several mission assets in its **Missions** array. Their array order defines the normal mission sequence.

Mission Information

Field	Purpose
Mission Id	Stable and unique identifier used by runtime state and save restoration.
Mission Title	User-facing mission title displayed by the UI.
Mission Description	Longer mission description displayed when the mission starts or in the journal.
Auto Start	Allows the mission to start when selected by the Mission Manager.
Auto Start Next Mission	Starts the following mission automatically after completion.

Do not change Mission IDs or Objective IDs after releasing a saved game unless you also provide a migration strategy. The Save System uses these stable identifiers to resolve restored progress even if array positions have changed.

Objective Order Mode

Ordered

Objectives are expected to progress in array order. Notifications that do not match the active objective are rejected. Use this mode for strongly directed sequences such as:

1. Find the key.
2. Unlock the door.
3. Enter the laboratory.

Flexible

Eligible progress for future objectives can be remembered and applied when those objectives become active. Use this mode when players may explore or perform actions in different orders.

Enable **Require Active Objective** on an individual objective when it must remain strictly ordered inside an otherwise Flexible mission.

Objective Types & Configuration

Objective Type	Typical Source	Important Configuration
ReachZone	MissionTriggerZone	Matching Objective ID and Player tag.

Objective Type	Typical Source	Important Configuration
<code>InteractWithTarget</code>	<code>MissionObjectiveNotifier</code> or legacy object notification	Objective ID and optional Target Object.
<code>CustomEvent</code>	<code>MissionObjectiveNotifier</code> , <code>UnityEvent</code> , or public API	Matching Objective ID.
<code>OwnItem</code>	RealFPS inventory	Target Item and Player Inventory reference.
<code>EquipItem</code>	RealFPS weapon system	Target Item and Weapon Manager reference.
<code>PickupItem</code>	<code>ItemPickup</code>	Target Item and optional Target Object.
<code>ToggleState</code>	<code>ToggleEventInteractable</code>	Objective ID and Required Toggle State.
<code>KillEnemy</code>	<code>EnemyHealth</code>	Optional Target Object and Required Kill Count.
<code>DeliverCargo</code>	<code>CarryableObject</code> and <code>MissionDeliveryZone</code>	Required Cargo ID and Required Delivery Count.

Common Objective Fields

Field	Purpose
Objective Id	Stable, case-sensitive identifier shared with scene notifiers.
Title	Short objective title displayed in the tracker and journal.
Description	Detailed player-facing instructions.
Objective Type	Selects the validation rule used by the Mission Manager.
Target Object	Optional scene object used by compatible objective types.
Target Item	<code>ItemData</code> required by inventory, pickup, or equipment objectives.
Hide Until Active	Prevents the objective from appearing before it becomes active.
Auto Complete	Completes the objective immediately when its condition is reached.
Require Active Objective	Prevents early validation, including in Flexible mode.
On Objective Started	Mission-specific <code>UnityEvent</code> invoked when the objective becomes active.
On Objective Completed	Mission-specific <code>UnityEvent</code> invoked when the objective completes.



Example — Starting a Computer

When a computer is controlled by `ToggleEventInteractable`, do not add a separate `MissionObjectiveNotifier` to its screen merely to detect the ON state. The toggle component already includes direct Mission System integration.

Configure the objective:

MissionData Field	Value
Objective Id	<code>start_computer</code>
Objective Type	<code>ToggleState</code>
Required Toggle State	Enabled
Auto Complete	Enabled
Target Object	The GameObject containing <code>ToggleEventInteractable</code>

Configure `ToggleEventInteractable` on the computer:

Toggle Field	Value
Mission Objective Id	<code>start_computer</code>
Notify Mission On State Changed	Enabled
Allow Interaction	Enabled
Use Toggle	Enabled
Initial Is On	Disabled

When the computer changes from OFF to ON, `ToggleEventInteractable` reports the new state directly to the Mission Manager. The objective completes only when the received state matches **Required Toggle State**.

Use `MissionObjectiveNotifier` when an object needs to validate an objective through a direct player interaction, an explicit `TriggerObjective()` call, or a custom UnityEvent. Do not duplicate notifications when a RealFPS component already provides native Mission System integration.

MissionManager & Notifications

`MissionManager` is the runtime authority of the Mission System. It selects the current mission and objective, accepts or rejects gameplay notifications, stores counters, invokes mission events, and refreshes the connected UI.

Use one primary Mission Manager per gameplay scene unless the project deliberately separates independent mission contexts. Explicit references are recommended if several managers exist because automatic scene searches cannot determine which manager is intended.

Mission Setup

Field	Purpose
Missions	Ordered array of <code>MissionData</code> assets available to this manager.
Starting Mission Index	Mission selected during automatic initialization.
Starting Objective Index	Objective selected during initialization or debug start.
Auto Initialize On Start	Initializes the configured starting mission automatically.

Array indexes are zero-based. Index `0` selects the first mission or objective.

Scene References

Reference	Required For	Automatic Search
Player Inventory	<code>OwnItem</code> objectives	Yes
Weapon Manager	<code>EquipItem</code> objectives	Yes
Mission Tracker UI	Current mission and objective HUD	Yes
Mission Popup UI	Mission and objective notifications	Yes
Mission Journal UI	Complete mission and objective list	Yes

The automatic search is convenient for a single-player scene. Assign references explicitly in complex scenes, additive scene setups, or scenes containing several candidate components.

Objective Lifecycle

The manager exposes three distinct objective moments:

Moment	Meaning
Started	The objective has become the current active objective.

Moment	Meaning
Reached	Its gameplay condition has been satisfied.
Completed	The objective has been finalized and progression may advance.

With **Auto Complete** enabled, Reached and Completed normally occur consecutively. With Auto Complete disabled, the objective can reach its condition while waiting for a script, animation, dialogue, or UnityEvent to call `CompleteCurrentObjective()`.

Global Events

`MissionManager` provides parameterless UnityEvents for general reactions and string events that include the relevant Mission ID or Objective ID.

Event Group	Examples of Use
Any Mission Started / Completed	Global music, analytics, or UI transitions.
All Missions Completed	End-game sequence or final reward.
Objective Started	Reveal a marker or enable a gameplay object.
Objective Reached	Play an animation or wait for confirmation before completion.
Objective Completed	Unlock the next area or hide an objective marker.
Events With Id	React only to a specific mission or objective through a listener.

Prefer `MissionObjectiveEventListener` when a scene object must react to one named objective. It avoids manually comparing IDs in several UnityEvent chains.



Choosing the Correct Notification Component

Gameplay Action	Recommended Source
Player enters an area	<code>MissionTriggerZone</code>
Player directly interacts with a custom target	<code>MissionObjectiveNotifier</code>
A custom UnityEvent completes an objective	Call <code>MissionObjectiveNotifier.TriggerObjective()</code>
A toggle changes state	Native <code>ToggleEventInteractable</code> mission fields
A pickup is collected	Native <code>ItemPickup</code> mission notification
An enemy is eliminated	Native enemy mission notification

Gameplay Action	Recommended Source
A locked door is opened	Native door mission notification
A destructible object is destroyed	<code>DestructibleMissionNotifier</code>
Cargo enters a delivery area	<code>CarryableObject</code> and <code>MissionDeliveryZone</code>

Use one authoritative notification path for each gameplay action. Do not add a generic notifier when the RealFPS component already contains Mission System integration.

MissionObjectiveNotifier

Use this component for direct interaction or custom event-driven objectives.

1. Add it to the interactable GameObject.
2. Enter an Objective ID that exactly matches `MissionData`.
3. Enable **Notify On Interact** when the Interaction Raycaster should trigger it.
4. Configure **Interaction Display Name** and **Interaction Verb** for the contextual prompt.
5. Enable **Trigger Only Once** for a one-shot objective.

`CanInteract()` returns false after a successful one-shot validation, so the Interaction Raycaster no longer displays a usable prompt.

For a UnityEvent-driven action, disable **Notify On Interact** and connect the external event to `TriggerObjective()`.

MissionTriggerZone

Use this component for `ReachZone` objectives.

- Its Collider is automatically configured as a trigger.
- **Required Tag** defaults to `Player`.
- **Trigger Only Once** becomes effective only after the Mission Manager accepts the notification.
- A rejected notification leaves the zone available for a later attempt.
- Scene gizmos make the zone visible during level design.

If the player has several colliders, ensure the collider entering the zone carries the required tag or belongs to the correctly tagged object expected by the project setup.

MissionObjectiveEventListener

This listener reacts to an event emitted by the Mission Manager without advancing mission progress itself.

Available modes:

- `ObjectiveStarted`
- `ObjectiveReached`
- `ObjectiveCompleted`

Set **Target Objective Id** to the exact ID to monitor, then connect **On Matched Objective Event** to the desired scene response. Enable **Trigger Only Once** for permanent one-shot reactions.

When **Accept Any Objective Id If Empty** is enabled and the target ID is empty, the listener reacts to every objective event of the selected mode.

A listener observes mission progression; a notifier requests mission progression. They serve opposite directions and should not be confused.

DestructibleMissionNotifier

Place this component on the same GameObject as `RealFPSDestructibleObject`. It subscribes to the destructible object's `On Destroyed` event and sends the configured Objective ID to the Mission Manager.

The component requires `RealFPSDestructibleObject`, automatically resolves local and scene references, and can prevent repeated validation with **Trigger Only Once**.

Debugging Missions

Mission Manager Inspector Buttons

During Play Mode, the Mission Manager can:

- start a selected mission and objective;

- restart the current objective;
- complete the current objective manually.

These controls are useful for testing later objectives without replaying the complete level.

MissionDebugStarter

Add `MissionDebugStarter` to a development-only `GameObject` when a test scene must begin at a specific mission and objective.

Field	Recommendation
Mission Index	Index of the mission to test.
Objective Index	Index of the objective to test.
Start On Start	Enable only in dedicated test scenes.
Preserve Existing Mission State	Keep enabled when Save System restoration may occur.
Start Delay	A small delay such as <code>0.1</code> seconds lets other systems initialize first.
Invoke Mission Start Events	Enable when the complete start flow must be tested.

When automatic debug start is enabled, the component disables the manager's normal automatic initialization before starting the selected objective. With **Preserve Existing Mission State** enabled, restored or already initialized progress is not overwritten.

Remove or disable debug starters before publishing a production scene.

Notification Troubleshooting

When a notification is refused, check these items in order:

1. A Mission Manager exists and contains the expected `MissionData` asset.
2. The mission has started.
3. The Objective ID matches exactly, including letter case.
4. The expected objective is active, or the mission uses Flexible order and permits future progress.
5. **Require Active Objective** is not blocking early completion.

6. The notifier has not already completed with **Trigger Only Once** enabled.
7. The native component's mission notification option is enabled.
8. Required target objects, items, cargo IDs, states, tags, and counters match the objective definition.
9. Debug logs are enabled on both the reporting component and Mission Manager.

An accepted notifier sets its runtime `Has Triggered` state. A refused notifier stays available, allowing the player to retry when the correct objective becomes active.

Delivery Cargo Objectives

`DeliverCargo` objectives count valid `CarryableObject` instances placed inside a `MissionDeliveryZone`. Each object contributes its configured delivery amount, and the Mission Manager completes the objective when the accumulated amount reaches the required count.

The official workflow uses:

- `CarryableObject` on every deliverable world object;
- `MissionDeliveryZone` on the destination trigger;
- a `DeliverCargo` objective inside `MissionData`.

No additional deliverable-item component is required.

How Cargo Counting Works

1. A `CarryableObject` enters the delivery zone.
2. The zone resolves the object's **Mission Cargo Id**.
3. The ID is compared with the objective's **Required Cargo Id**.
4. A matching object is added to the zone's unique counted-object set.
5. Its **Mission Delivery Amount** is added to the current cargo total.
6. The zone sends the complete current total to the Mission Manager.
7. The objective is reached when the total is greater than or equal to **Required Delivery Count**.

One object is counted once, even when it contains several colliders. Leaving the zone removes its contribution; entering again allows it to be counted again unless the current objective transition rules temporarily block it.



Preparing a Cargo Object

Add or configure `CarryableObject` on the crate, container, supply box, or other physical object that the player must deliver.

In the **Mission Cargo** section, configure:

Field	Purpose	Example
Mission Cargo Id	Category used by delivery objectives. Comparison is case-insensitive.	<code>supply_crate</code>
Mission Delivery Amount	Amount contributed by this object. Minimum value is <code>1</code> .	<code>1</code>

The object also needs the normal Carryable Object physics configuration, including an appropriate Collider and Rigidbody. The delivery zone resolves `CarryableObject` from the entering collider or one of its parents.

Use the same Cargo ID for interchangeable objects. Use different IDs when a mission must distinguish supplies, medical crates, fuel containers, or other cargo categories.

Weighted Cargo

`Mission Delivery Amount` can represent more than one unit. For example:

Object	Cargo ID	Delivery Amount
Small supply box	<code>supplies</code>	<code>1</code>
Large supply crate	<code>supplies</code>	<code>3</code>
Supply pallet	<code>supplies</code>	<code>5</code>

An objective requiring `5` units can therefore be completed with five small boxes, one pallet, or any valid combination whose total reaches five.



Configuring the Mission Objective

Create an objective in `MissionData` with these values:

Field	Example Value
Objective Id	<code>deliver_supplies</code>
Title	<code>Deliver the Supplies</code>
Description	<code>Place five units of supplies inside the marked area.</code>
Objective Type	<code>DeliverCargo</code>
Required Cargo Id	<code>supplies</code>
Required Delivery Count	<code>5</code>
Auto Complete	Enabled

`Target Object` is not used by DeliverCargo objectives. The connection is made through the Objective ID and Cargo ID.

Use a unique Objective ID across all missions assigned to the same Mission Manager. When **Use Objective Data** is enabled, the delivery zone searches the manager's mission list for a matching DeliverCargo objective and uses the first valid match.

Creating the Delivery Zone

Create a zone through:

Tools/BLInformatique/Real FPS/Addons/  Mission System/Create/  Mission Delivery Zone

Then configure:

Field	Recommended Value
Objective Id	Exact objective ID, for example <code>deliver_supplies</code>
Use Objective Data	Enabled
Require Tag	Disabled unless the project has a dedicated cargo tag workflow
Require Fresh Entry After Objective Activation	Enabled
Force Refresh On Objective Change	Enabled
Enable Periodic Rescan	Enabled
Rescan Interval	<code>0.25</code> seconds

Scale its trigger Collider to cover the complete destination area. Avoid placing the trigger where unrelated carried objects regularly pass through it.

Objective Data and Fallback Settings

With **Use Objective Data** enabled, the zone reads these values from the matching `MissionData` objective:

- Required Cargo Id;
- Required Delivery Count.

The local fallback Cargo ID is used only when matching objective data cannot be resolved or **Use Objective Data** is disabled.

For production missions, prefer Objective Data. It keeps the mission requirements in one authoritative asset and reduces configuration mismatches between the zone and MissionData.

Fresh Entry After Activation

When **Require Fresh Entry After Objective Activation** is enabled, valid cargo already inside the zone when the objective becomes active is temporarily blocked. The object must:

1. leave the zone;
2. enter the zone again.

This prevents players from pre-positioning cargo before the mission begins and receiving immediate completion when the objective activates.

Disable this option when pre-delivered cargo should count immediately.

Automatic Refresh and Rescan

Force Refresh On Objective Change rebuilds the zone state when the active mission or objective changes. This is important when the same zone is reused across several objectives.

Enable Periodic Rescan checks the physical overlaps at runtime. It keeps the zone synchronized if normal trigger messages are missed because an object is enabled, disabled, teleported, restored, or moved during initialization.

The default interval of 0.25 seconds is appropriate for normal delivery areas. Increase it only when profiling shows that many large zones create unnecessary physics queries.

Delivery Zone Events

Event	Invoked When
On Cargo Entered	A valid object starts contributing to the total.
On Cargo Exited	A previously counted object stops contributing.
On Cargo Rejected	An entering object fails a configured acceptance rule.

These events can drive sounds, lights, barriers, animations, or feedback messages. They do not replace mission validation; the zone continues to report the calculated cargo total to the Mission Manager.

Runtime Diagnostics

During Play Mode, inspect the following read-only values:

Runtime Field	Meaning
Runtime Cargo Count	Sum of all accepted delivery amounts.
Runtime Object Count	Number of unique accepted Carryable Objects.
Runtime Resolved Cargo Id	Cargo ID currently required by the zone.
Runtime Resolved Required Count	Total required by the resolved objective.
Runtime Using Objective Data	Confirms whether MissionData was resolved successfully.
Runtime Last Mission / Objective	Last active mission state observed by the zone.
Runtime Blocked Fresh Entry Count	Objects waiting to exit and enter again.

Troubleshooting a Rejected Crate

Check these items in order:

1. The object contains `CarryableObject` on the collider or a parent.
2. The Cargo ID matches the objective's Required Cargo ID.
3. The zone Objective ID exactly matches the DeliverCargo Objective ID.
4. **Runtime Using Objective Data** is enabled during Play Mode.
5. The active objective is the expected DeliverCargo objective, or Flexible progression permits it.
6. The object is not listed in **Runtime Blocked Fresh Entry Count**.
7. A required tag, when enabled, exists on the Carryable Object GameObject.
8. The object and zone have a valid Unity physics setup.
9. Debug logs are enabled on both the zone and Mission Manager.

If the object is already inside when the objective activates, remove it completely from the trigger and place it inside again before treating the configuration as faulty.

Flexible Mission Order

In a Flexible mission, delivery progress can be remembered for a future eligible objective. The Mission Manager restores the remembered delivery count when that objective becomes active.

Enable **Require Active Objective** on the DeliverCargo objective when cargo must never be delivered early.



Mission User Interfaces

The Mission System provides three complementary player-facing interfaces. They are optional and can be used independently.

Interface	Purpose	Recommended Use
<code>MissionTrackerUI</code>	Permanently displays the current mission, objective, icon, and progress.	Compact HUD element.
<code>MissionPopupUI</code>	Temporarily announces mission and objective changes.	Short animated notifications.
<code>MissionJournalUI</code>	Displays the current mission and its complete objective list.	Player-opened journal or pause-style overlay.

`MissionObjectiveEntryUI` is the reusable row prefab instantiated by the journal. It is not normally placed directly in the gameplay HUD.

None of these interfaces controls mission progression. They display the state owned by `MissionManager`.

MissionTrackerUI

The tracker is the primary mission HUD. Assign it to **Mission Tracker UI** on the Mission Manager, or leave the manager reference empty when a single tracker can be found automatically in the scene.

Required References

Reference	Purpose
Root Panel	Optional parent object shown or hidden with the tracker.
Canvas Group	Controls visible and hidden alpha values.
Tracker Icon	Displays an icon matching the active objective type.
Mission Title Text	Displays the current mission title.
Objective Title Text	Displays the active objective title.
Objective Description Text	Displays the active objective description.
Progress Text	Displays counters for supported objective types.

Objective Icons

Assign sprites for Reach Zone, Interact, Item, Equip, Toggle, Kill, Deliver, and Custom Event objectives. **Default Icon** is used when a specific icon is unavailable.

Visibility

- **Hide When No Mission** hides the tracker when no mission is active.
- **Hide Progress When Empty** hides the progress text object when the active objective has no counter.
- **Visible Alpha** and **Hidden Alpha** control the Canvas Group states.

Kill and delivery objectives display their current count and required total. Other objective types normally leave the progress field empty.

MissionPopupUI

The popup displays short messages for mission lifecycle events. Assign it to **Mission Popup UI** on the Mission Manager.

The manager can request these popup types:

- Mission Started;
- Objective Updated;
- Objective Completed;
- Mission Completed;
- Mission Failed.

Animation Settings

Field	Purpose
Fade In Duration	Time used to reveal the popup.
Hold Duration	Time the popup remains fully visible.
Fade Out Duration	Time used to hide the popup.
Hide Object When Inactive	Disables the Root Panel after the animation.

Assign a Canvas Group for fading. Without one, the message is still displayed for the hold duration and the Root Panel can still be hidden afterward.

Each popup category can use its own icon and label. Keep labels short because the mission or objective title is displayed separately as the message.

Popup animation currently uses scaled game time. If another system sets `Time.timeScale` to zero while a popup is visible, its animation waits until scaled time resumes.

MissionJournalUI

The journal displays the current mission title, description, and a generated list of objectives. Assign it to **Mission Journal UI** on the Mission Manager.

Required References

Reference	Purpose
Root Panel	Complete journal panel shown or hidden at runtime.
Journal Header Text	Displays the configurable journal heading.
Mission Title Text	Displays the current mission title.
Mission Description Text	Displays the current mission description.
Objectives Container	Parent Transform receiving generated objective rows.
Objective Entry Prefab	<code>MissionObjectiveEntryUI</code> prefab instantiated for each objective.

Enable **Start Hidden** for a journal opened by the player. The default old-input shortcut is `J`.

Input Systems

Old Input System

1. Enable **Use Old Input**.
2. Set **Toggle Journal Key**.
3. Disable **Use New Input** unless both paths are deliberately supported.

New Input System

1. Enable **Use New Input**.
2. Assign **Toggle Journal Action**.
3. Ensure the action is compatible with a button press.

The journal enables and disables its assigned Input Action with the component lifecycle.

Gameplay Behaviour

Opening the journal refreshes its content and activates the Root Panel. Closing it hides the panel.

The journal is a read-only mission display. By itself it does not:

- pause gameplay;
- change `Time.timeScale` ;
- unlock or display the cursor;
- disable Player input.

This behaviour is appropriate for a quick keyboard overlay. If the journal is redesigned with clickable buttons, combine it with the project's normal UI freeze and cursor-management workflow.

MissionObjectiveEntryUI

Create a prefab containing this component and assign it to the journal's **Objective Entry Prefab** field.

Each generated row can display:

- objective title;
- state label;
- state icon;
- background color.

Supported visual states are Locked, Active, Completed, and Failed. Configure an icon, foreground color, background color, and label for each state.

The journal rebuilds its generated rows when refreshed. Do not manually place persistent objective rows inside the Objectives Container; use the assigned prefab instead.



Recommended Canvas Structure

```

MissionCanvas
├── MissionTracker
│   ├── ObjectiveIcon
│   ├── MissionTitle
│   ├── ObjectiveTitle
│   ├── ObjectiveDescription
│   └── Progress
├── MissionPopup
│   ├── PopupIcon
│   ├── PopupState
│   └── PopupMessage
└── MissionJournal
    ├── JournalHeader
    ├── MissionTitle
    ├── MissionDescription
    ├── ObjectivesContainer
    └── ObjectiveEntry instances

```

The three panels can share one Canvas or use separate canvases according to the project's sorting and animation requirements.

Connecting the Interfaces

1. Create and configure the desired UI panels.
2. Assign the Tracker, Popup, and Journal components to the corresponding Mission Manager fields.
3. Enter Play Mode.
4. Start a mission and confirm that the tracker displays the active objective.
5. Complete an objective and confirm that the popup appears.
6. Press the journal shortcut and confirm that all objective rows are generated.

The Mission Manager also searches the scene automatically when these references are empty. Explicit assignment is recommended for predictable production scenes.

UI Troubleshooting

Tracker remains hidden

- Confirm that a mission has started.
- Check **Hide When No Mission**.
- Verify Root Panel and Canvas Group references.
- Confirm that visible alpha is greater than zero.

Popup never appears

- Assign the Popup UI to the Mission Manager.
- Verify that the Root Panel is not controlled by another script.
- Check the Canvas sorting order and alpha.
- Confirm that game time is not permanently frozen during the animation.

Journal opens but contains no objectives

- Assign the Mission Manager.
- Assign Objectives Container and Objective Entry Prefab.
- Confirm that the current MissionData contains objectives.
- Check that the entry prefab contains `MissionObjectiveEntryUI` and valid TMP references.

Journal shortcut does not work

- Enable the intended input path.
- For the old system, confirm the configured KeyCode.
- For the new system, assign a valid Input Action and confirm that the Input System package and scripting define are active.
- Avoid binding both input paths to the same physical key unless duplicate toggles have been tested.



Save System Integration



Save System Integration

RealFPS Mission System can integrate with the optional **RealFPS Save System** addon. The Mission System does not require Save System to run, and the bridge uses optional runtime discovery so that neither package creates an invalid hard dependency when the other addon is absent.

RealFPS Save System is a separate addon and is not included with RealFPS Mission System.

Requirements

Before configuring mission persistence:

1. Verify that RealFPS Core and Mission System work correctly without saving.
2. Import RealFPS Save System.

3. Open **Tools/BLInformatique/Real FPS/Setup Center**.
4. Run the Save System and Mission System setup or repair operations.
5. Let Unity finish compiling and confirm that the Console contains no red errors.

Test the complete mission flow before adding persistence. Saving cannot correct an invalid MissionData configuration or a notification that never reaches the Mission Manager.

What Is Saved

Mission Manager State

`SaveableMissionManager` captures:

- current mission index;
- current objective index;
- stable Mission ID and Objective ID;
- mission started state;
- all-missions-completed state;
- current kill count;
- current delivery count;
- remembered Flexible objective progress.

Stable IDs allow restoration to resolve the intended mission and objective even when array indexes have changed. Do not rename released Mission IDs or Objective IDs without a save migration strategy.

One-Shot Scene State

The optional bridges can also preserve:

- `MissionTriggerZone.Has Triggered ;`
- `MissionObjectiveNotifier.Has Triggered .`

This prevents successfully validated one-shot zones and notifiers from becoming available again after loading.



Saving the MissionManager

On the same `GameObject` as `MissionManager`, add:

1. `RealFPSSaveableEntity`;
2. `SaveableMissionManager`.

`SaveableMissionManager` requires `RealFPSSaveableEntity` and normally assigns both references automatically.

Verify these fields:

Field	Expected Reference
Saveable Entity	The local <code>RealFPSSaveableEntity</code> .
Mission Manager Component	The local <code>MissionManager</code> .

Every saveable entity must have a stable, unique Save ID. Do not duplicate the Mission Manager `GameObject` after assigning its final identity without regenerating or verifying the resulting ID.

Restore Timing

Mission state is applied immediately for restoration inside an already running scene. It is applied once more on the following frame to protect scene-load restoration from being overwritten by `MissionManager.Start()` or automatic initialization.

After restored zones and notifiers have finished, the bridge refreshes the mission UI again at the end of the frame.

This delayed second pass is intentional. Do not remove it merely because the first restore appears to work in an in-game test.



Saving MissionTriggerZone

Add these components to each one-shot trigger whose runtime state must survive loading:

1. `MissionTriggerZone` ;
2. `RealFPSSaveableEntity` ;
3. `SaveableMissionTriggerZone` .

The bridge automatically locates the local Mission Trigger Zone and saves its `Has Triggered` value.

A trigger does not need a save bridge when repeating after load is acceptable or when the restored Mission Manager state already makes the repeated trigger harmless. Add saveables only where persistent one-shot behaviour matters.



Saving MissionObjectiveNotifier

Add these components to each one-shot notifier whose runtime state must survive loading:

1. `MissionObjectiveNotifier` ;
2. `RealFPSSaveableEntity` ;
3. `SaveableMissionObjectiveNotifier` .

The bridge restores `Has Triggered` . With **Trigger Only Once** enabled, the notifier remains unavailable and its Interaction Raycaster prompt stays hidden after restoration.



Delivery Progress and Scene Objects

The Mission Manager saves the current delivery counter and remembered Flexible delivery progress. `MissionDeliveryZone` also recalculates its live total from overlapping `CarryableObject` instances.

For a reliable restored delivery scene:

- save the transforms and active states of relevant cargo objects with the appropriate Save System components;
- keep Cargo IDs stable;
- keep the delivery Objective ID stable;
- allow the zone's automatic refresh and periodic rescan to rebuild its physical overlap state.

The delivery zone itself does not require a dedicated mission save bridge. Its physical count is derived from restored cargo placement, while mission progression is restored by `SaveableMissionManager` .



Recommended Save/Load Test

Use a dedicated test mission containing at least three objectives:

1. a one-shot Trigger Zone;
2. a multi-count Kill or Delivery objective;
3. a one-shot Mission Objective Notifier.

Then test:

1. Start a new game.
2. Complete the Trigger Zone objective.
3. Make partial progress on the counter objective.
4. Save the game.
5. Continue and change the mission state.
6. Load the previous save without leaving Play Mode.
7. Confirm the restored mission, objective, and counter.
8. Confirm that the completed trigger remains unavailable.
9. Confirm that UI text and progress are refreshed.
10. Repeat the same test by leaving the scene or application and loading the slot again.

Both restoration paths must be tested because in-session restoration and scene-load restoration have different initialization timing.



Save Integration Troubleshooting

Mission returns to the first objective

- Confirm that `SaveableMissionManager` is on the Mission Manager GameObject.
- Verify that its Saveable Entity has a valid unique ID.
- Confirm that Mission IDs and Objective IDs have not been renamed.
- Check that automatic debug starters are not overwriting restored state.
- Keep **Preserve Existing Mission State** enabled on development debug starters.

Correct objective but wrong counter

- Confirm that the objective type still matches the saved objective.
- Verify Required Kill Count or Required Delivery Count.
- For deliveries, confirm that restored cargo positions agree with the saved mission counter.
- Keep Delivery Zone rescan enabled during diagnosis.

Trigger or notifier becomes usable again

- Add its dedicated `SaveableMissionTriggerZone` or `SaveableMissionObjectiveNotifier` bridge.
- Verify the local `RealFPSSaveableEntity` and unique Save ID.
- Confirm that the bridge resolved the correct local mission component.

UI remains stale after loading

- Assign Tracker, Popup, and Journal references to the Mission Manager.
- Confirm that `SaveableMissionManager` is present and enabled.
- Enable **Verbose Logs** temporarily and look for the post-restore UI refresh.
- Check that another UI script is not overwriting the restored text on the following frame.

Bridge reports that Mission System is unavailable

- Run Setup Center again.
- Check compilation errors in both addons.
- Verify that the Mission System assembly and scripts are present.
- Do not manually create direct Save System references inside Mission System Core scripts.



General Troubleshooting

Use this checklist after completing the component-specific troubleshooting steps in the previous chapters.

Mission Does Not Start

1. Confirm that a `MissionManager` exists and is enabled.
2. Verify that its **Missions** array contains a valid `MissionData` asset.

3. Check **Starting Mission Index** and **Starting Objective Index**.
4. Enable **Auto Initialize On Start**, or start the mission explicitly through script or debug controls.
5. Confirm that the selected mission contains at least one valid objective.
6. Disable development `MissionDebugStarter` components that may override the intended start.

Objective Notification Is Refused

- Compare Objective IDs character by character; IDs are case-sensitive.
- Confirm that the mission has started.
- Check whether the expected objective is active.
- In Flexible mode, verify that **Require Active Objective** is disabled when early progress should be remembered.
- Confirm that the notifying component has not already triggered once.
- Enable debug logs on the notifier and Mission Manager.

Wrong Objective Completes

- Give every objective a unique ID across all missions assigned to the same manager.
- Remove duplicate notification paths from the same gameplay object.
- Do not combine a native RealFPS mission integration with a generic notifier unless both actions are intentionally different.
- Verify Target Object, Target Item, Cargo ID, toggle state, and enemy filters.

Counter Never Reaches Its Target

- Check Required Kill Count or Required Delivery Count.
- Confirm that each enemy or cargo object reports the intended amount.
- For cargo, inspect Runtime Cargo Count and Runtime Object Count.
- Verify that repeated notifications are accepted by the active objective.
- Check whether a Flexible future objective is storing progress rather than displaying it immediately.

Objective Reaches Its Condition but Does Not Advance

- Check whether **Auto Complete** is disabled.
- If disabled intentionally, call `MissionManager.CompleteCurrentObjective()` after the dialogue, animation, confirmation, or reward sequence finishes.
- Observe Objective Reached and Objective Completed as separate events.

Next Mission Does Not Start

- Enable **Auto Start Next Mission** on the completed MissionData.
- Verify that another valid mission exists after it in the manager's Missions array.
- Check for a null mission entry.
- Confirm that no completion UnityEvent disables the Mission Manager.

UI Is Empty or Stale

- Assign Tracker, Popup, and Journal components to the Mission Manager.
- Verify all TMP references and Root Panels.
- Confirm that the mission is actually initialized.
- After restoration, verify that `SaveableMissionManager` performs its post-restore refresh.
- Check for another script clearing or disabling the same UI hierarchy.

Input or Interaction Prompt Does Not Work

- Confirm that the Player contains a configured `InteractionRaycaster`.
- Check the ray distance, interaction layer mask, and blockers.
- Verify that the target Collider can be hit by the ray.
- Confirm that `MissionObjectiveNotifier.Notify On Interact` is enabled.
- Configure Interaction Display Name and Interaction Verb.
- Do not expect a notifier on an unrelated child UI object to react to a toggle on its parent.

Build Fails with Editor Namespace Errors

- Ensure runtime scripts do not import `UnityEditor` outside `#if UNITY_EDITOR`.
- Keep Editor-only menu scripts inside an Editor folder or Editor-only assembly.
- Reimport the package and clear all Console compilation errors before building.

Setup Center Does Not Detect an Addon

- Confirm that the addon remains in its original RealFPS folder structure.
- Let Unity finish compiling before running setup.
- Fix all compilation errors first.
- Run the setup or repair operation again after adding or removing an addon.

- Do not manually force optional scripting defines while the related addon is absent.

Clean Diagnostic Procedure

When a complex mission fails, reproduce it in a minimal scene containing only:

- one configured RealFPS Player;
- one Mission Manager;
- one MissionData asset;
- one objective;
- one matching notifier or zone;
- optional Tracker UI;
- debug logs enabled.

After the minimal version works, reintroduce Save System, UI animation, additional objectives, and scene-specific logic one system at a time.

? Frequently Asked Questions

Is RealFPS Core required?

Yes. Mission System is an addon for RealFPS Core and is not a standalone mission framework.

Is RealFPS Save System required?

No. Missions work without persistence. Save System is required only when mission progress must survive save/load operations.

Must I use every supplied UI component?

No. Tracker, Popup, and Journal are independent. Use only the interfaces needed by the project.

Can I create missions without code?

Yes. Most workflows use MissionData, Inspector configuration, ready-made notifiers, and UnityEvents. Custom scripts are useful only for project-specific gameplay actions.

Are Objective IDs case-sensitive?

Yes. `open_door` and `Open_Door` are different identifiers. Copy IDs exactly and avoid spaces.

Can objectives be completed out of order?

Yes. Set **Objective Order Mode** to `Flexible`. Eligible future progress is remembered and applied when the objective becomes active.

Can one objective remain strictly ordered inside a Flexible mission?

Yes. Enable **Require Active Objective** on that objective.

What is the difference between Reached and Completed?

Reached means the gameplay condition has been satisfied. Completed means progression has been finalized. They normally occur consecutively when **Auto Complete** is enabled.

How do I wait for a dialogue or animation before advancing?

Disable **Auto Complete**. React to the Objective Reached event, play the required sequence, then call `CompleteCurrentObjective()`.

Should I add MissionObjectiveNotifier to every interactive object?

No. Many RealFPS components already report mission events directly, including toggles, pickups, enemies, and doors. Use the generic notifier only for custom interactions or explicit UnityEvent calls.

How do I complete an objective when a computer turns on?

Use a `ToggleState` objective. Enter the same Mission Objective ID in `ToggleEventInteractable` and enable **Notify Mission On State Changed**. A separate notifier on the computer screen is unnecessary.

Can several objects contribute to one objective?

Yes. Kill and DeliverCargo objectives use counters. Multiple matching enemies or cargo objects can contribute to the required total.

Can one cargo object count as several units?

Yes. Set **Mission Delivery Amount** on `CarryableObject` to a value greater than one.

Why is cargo already inside the zone not counted?

When **Require Fresh Entry After Objective Activation** is enabled, the object must leave and enter again after the objective becomes active.

Can several Mission Managers exist in one scene?

Technically yes, but automatic searches cannot know which one is intended. Assign every reference explicitly and keep the mission contexts completely separate.

Can I rename Mission IDs or Objective IDs later?

During development, yes, if every reference is updated. After releasing saved games, changing stable IDs can prevent old saves from resolving the intended mission state.

Does the Journal pause gameplay?

No. The supplied Journal is a read-only overlay. Add the project's normal UI freeze and cursor workflow if a pause-style interactive journal is required.

Why does a popup stop while the game is paused?

Popup animation uses scaled game time. It resumes when `Time.timeScale` becomes greater than zero.

How should I test mission saving?

Test both loading inside an active Play session and loading after a scene or application restart. These restoration paths use different initialization timing.

Compatibility

Unity Version

This version of RealFPS Mission System is authored and documented for:

Unity 6000.6.0f1

Compatible Unity 6 maintenance releases may work, but the package should be validated again after changing Unity versions.

Required Packages and Assets

Dependency	Status	Purpose
RealFPS Core	Required	Player, inventory, weapons, interaction, enemies, doors, toggles, and shared runtime APIs.
TextMeshPro	Required for supplied UI	Tracker, Popup, Journal, and objective entry text.
RealFPS Save System	Optional	Mission and one-shot notifier persistence.
RealFPS Moving Platform System	Optional	Platform-driven mission events when its bridge is installed.

Render Pipelines

Mission logic is not tied to a specific render pipeline. The runtime mission system can operate independently from Built-in, URP, or HDRP rendering.

The supplied RealFPS project, UI assets, materials, demo scenes, and visual effects may have their own render-pipeline requirements. Follow the compatibility information supplied with RealFPS Core and the demo content.

Input Systems

Mission gameplay notifications are independent from the selected input backend.

The supplied Journal supports:

- the legacy Unity Input Manager through a `KeyCode` ;
- the new Input System through an assigned `InputAction` when the package and scripting define are available.

Configure only the input paths used by the project and test duplicate bindings when both are enabled.

Platforms

The Mission System uses standard Unity runtime APIs and contains no intended desktop-only mission logic. Actual platform support also depends on RealFPS Core, the selected input setup, UI configuration, Save System storage, and any third-party integrations used by the project.

Always create a Development Build for each target platform before release.

Optional Addon Isolation

RealFPS Core does not require Mission System. Mission System does not require Save System or Moving Platform System. Optional integrations use bridges and setup defines so the packages can compile in

supported combinations.

Run Setup Center after installing, updating, or removing an addon, and validate these combinations when preparing a release:

1. RealFPS Core only;
2. Core + Mission System;
3. Core + Mission System + Save System;
4. Core + Mission System + Moving Platform System;
5. all installed addons used by the project.

Build Validation

Before publishing:

- test an Editor Play session;
- create a Development Build;
- confirm that runtime assemblies contain no unguarded `UnityEditor` references;
- test both old and new input configurations used by the package;
- test mission restoration when Save System is included;
- test a clean project import with no optional addon installed.

Conclusion & Support

RealFPS Mission System is designed to keep mission content modular, readable, and connected to real gameplay actions without forcing project-specific logic into one large manager.

For reliable production missions:

1. use stable and unique Mission IDs and Objective IDs;
2. test one objective type at a time;
3. use native RealFPS mission integrations when available;
4. keep one authoritative notification path per action;
5. validate the mission flow before adding persistence;
6. test Save System restoration both in-session and after scene loading;

7. use debug tools only in development scenes;

8. create a Development Build before release.

Recommended Final Checklist

- MissionData assets contain no duplicate IDs.
- Mission Manager arrays contain no null entries.
- All scene notifiers use the intended manager and exact Objective ID.
- Ordered and Flexible behaviours have been tested deliberately.
- Tracker, Popup, and Journal show the correct state.
- One-shot trigger and notifier states restore correctly when Save System is installed.
- Delivery zones rebuild the expected cargo count.
- Debug logs and development starters are disabled where appropriate.
- Setup Center reports the installed addons correctly.
- The target platform build contains no compilation errors.

Support

Website: <https://www.blinformatique.fr>

Email: realfps@blinformatique.fr

When requesting support, include:

- Unity version;
- RealFPS and addon versions;
- target platform;
- complete Console error or warning;
- relevant Inspector screenshots;
- exact reproduction steps;
- whether the issue occurs in Editor, Development Build, or both.

Clear reproduction information helps distinguish mission configuration errors from integration, physics, UI, input, or save-restoration issues.

CONTENTS

 Filter components...

-  Table of Contents
-  Overview
-  How To Use
-  Installation & Quick Start
-  Mission Architecture & MissionData
-  Objective Types & Configuration
-  MissionManager & Notifications
-  Debugging Missions
-  Delivery Cargo Objectives
-  Mission User Interfaces
-  Save System Integration
-  General Troubleshooting
-  Frequently Asked Questions
-  Compatibility
-  Conclusion & Support
-  Components (12)

BLINFORMATIQUE.REALFPS

- MissionData
- MissionDebugStarter
- MissionDeliveryZone
- MissionJournalUI
- MissionManager
- MissionObjectiveEntryUI
- MissionObjectiveEventListener

MissionObjectiveNotifier

MissionPopupUI

MissionTrackerUI

MissionTriggerZone

BLINFORMATIQUE.REALFPS.MISSIONS

DestructibleMissionNotifier

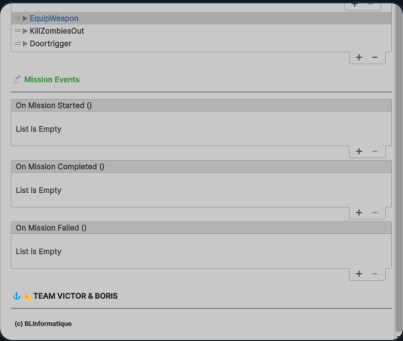
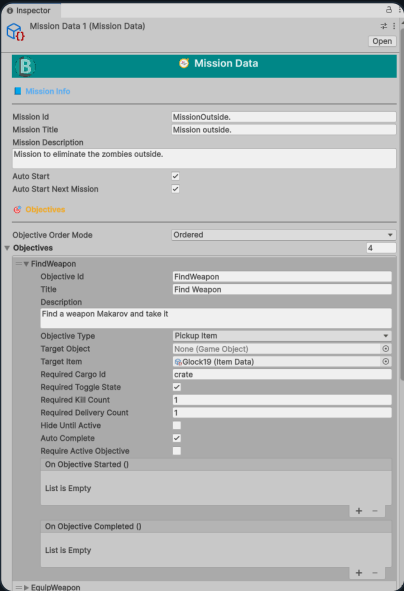


Components

Tooltips are in **English**. Missing tooltips can be completed automatically from field names. Tables list documented fields. Optional notes (Markdown) and images are shown below each component when present.

MissionData
 ScriptableObject

BLInformatique.RealFPS



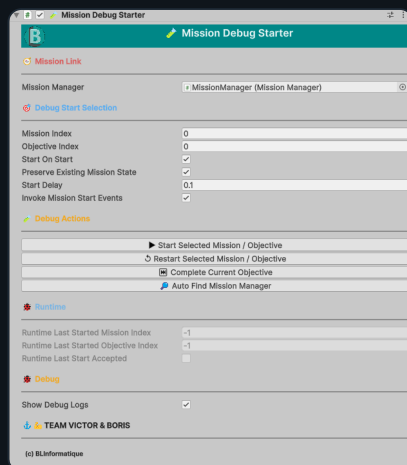
Field	Type	Tooltip	Hints
missionId	String	Unique identifier for this mission.	—
missionTitle	String	Display title shown in the mission UI.	—
missionDescription	String	Description shown when the mission starts. • TextArea	TextArea
autoStart	Boolean	If true, the mission starts automatically when assigned to MissionManager.	—
autoStartNextMission	Boolean	If true, the next mission is started automatically when this one is completed.	—

Field	Type	Tooltip	Hints
↓			
objectiveOrderMode	ObjectiveOrderMode	Defines whether objectives must be completed in order or whether future objective progress can be remembered.	Enum
objectives	MissionObjectiveData[]	Ordered list of objectives for this mission.	List / Array
↓			
onMissionStarted	UnityEvent	Invoked when the mission starts.	UnityEvent
onMissionCompleted	UnityEvent	Invoked when the mission is completed.	UnityEvent
onMissionFailed	UnityEvent	Invoked when the mission fails.	UnityEvent

MissionDebugStarter MonoBehaviour

BLInformatique.RealFPS

AddComponentMenu: BLInformatique/Real FPS/Missions/ Mission Debug Starter



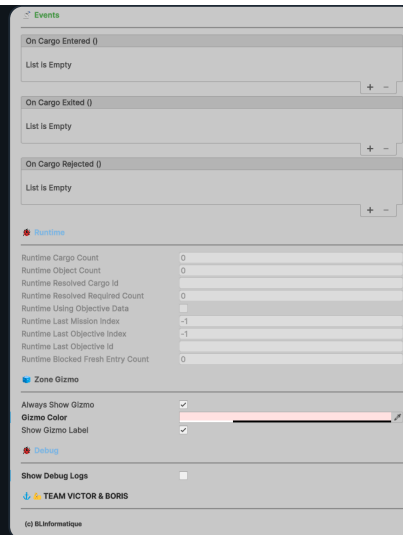
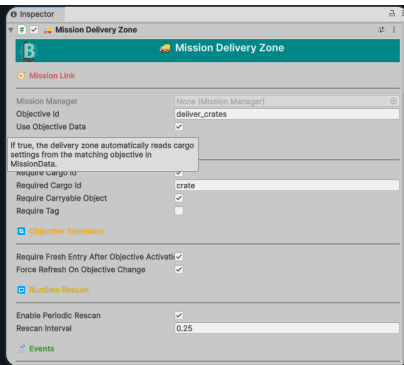
Field	Type	Tooltip	Hints
↓			
missionManager	MissionManager	MissionManager used to start a mission at a selected objective. If empty, one will be searched automatically.	Object Reference
↓			
missionIndex	Int32	Mission index to start.	—
objectiveIndex	Int32	Objective index to start inside the selected mission.	—

Field	Type	Tooltip	Hints
startOnStart	Boolean	If true, this component starts the selected mission and objective automatically on Start.	—
preserveExistingMissionState	Boolean	If true, automatic debug start is skipped when the MissionManager already contains initialized or restored runtime state.	—
startDelay	Single	Delay in seconds before starting the selected mission objective on Start.	—
invokeMissionStartEvents	Boolean	If true, mission started events and popups are invoked when using this debug starter.	—
↓			
startSelectedMissionObjectiveButton	Boolean	Starts the selected mission and objective.	—
restartSelectedMissionObjectiveButton	Boolean	Restarts the selected mission and objective.	—
completeCurrentObjectiveButton	Boolean	Completes the current objective on the linked MissionManager.	—
autoFindMissionManagerButton	Boolean	Searches the scene for a MissionManager.	—
↓			
runtimeLastStartedMissionIndex	Int32	Last mission index started by this debug starter.	—
runtimeLastStartedObjectiveIndex	Int32	Last objective index started by this debug starter.	—
runtimeLastStartAccepted	Boolean	True if the last start request was accepted.	—
↓			
showDebugLogs	Boolean	If true, prints debug logs in the Console.	—

MissionDeliveryZone MonoBehaviour

BLInformatique.RealFPS

AddComponentMenu: BLInformatique/Real FPS/Missions/ Mission Delivery Zone • Requires: Collider



Field	Type	Tooltip	Hints
↓			
missionManager	MissionManager	MissionManager receiving delivery notifications. If null, one will be searched in the scene.	Object Reference
objectiveId	String	Objective identifier expected by the MissionManager.	—
useObjectiveData	Boolean	If true, the delivery zone automatically reads cargo settings from the matching objective in MissionData.	—
↓			
requireCargold	Boolean	If true, only objects with a matching mission cargo ID are counted when fallback data is used.	—
requiredCargold	String	Fallback cargo identifier used when no objective data can be resolved.	—
requireTag	Boolean	If true, this zone accepts objects tagged with the required tag.	—
requiredTag	String	Required tag for colliders entering the zone.	—
↓			
requireFreshEntryAfterObjectiveActivation	Boolean	If true, cargo already inside the zone when the objective becomes active is ignored until it exits and enters again.	—

Field	Type	Tooltip	Hints
forceRefreshOnObjectiveChange	Boolean	If true, the zone automatically refreshes when the active mission objective changes.	—
↓			
enablePeriodicRescan	Boolean	If true, the zone periodically rescans overlapping colliders to stay synchronized.	—
rescanInterval	Single	Delay in seconds between automatic trigger rescans.	—
↓			
onCargoEntered	UnityEvent	Invoked when a valid cargo object starts being counted.	UnityEvent
onCargoExited	UnityEvent	Invoked when a valid cargo object stops being counted.	UnityEvent
onCargoRejected	UnityEvent	Invoked when a collider enters but is rejected.	UnityEvent
↓			
runtimeCargoCount	Int32	Current counted cargo amount inside this zone.	—
runtimeObjectCount	Int32	Current number of counted carryable objects inside this zone.	—
runtimeResolvedCargoId	String	Resolved cargo identifier currently used by the zone.	—
runtimeResolvedRequiredCount	Int32	Resolved delivery amount currently required by the active objective.	—
runtimeUsingObjectiveData	Boolean	True if objective data was successfully resolved from MissionData.	—
runtimeLastMissionIndex	Int32	Last active mission index seen by this zone.	—
runtimeLastObjectiveIndex	Int32	Last active objective index seen by this zone.	—
runtimeLastObjectiveId	String	Last active objective identifier seen by this zone.	—
runtimeBlockedFreshEntryCount	Int32	Number of blocked carryable objects waiting for a fresh re-entry.	—

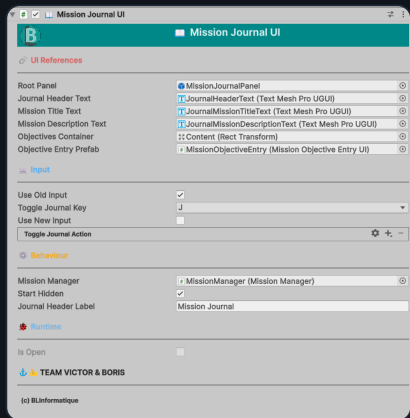
Field	Type	Tooltip	Hints
↓			
alwaysShowGizmo	Boolean	If true, the trigger gizmo is always visible in the Scene view.	—
gizmoColor	Color	Color of the trigger gizmo.	—
showGizmoLabel	Boolean	If true, a label is displayed in the Scene view for this zone (Editor only).	—
↓			
showDebugLogs	Boolean	If true, prints debug logs in the Console.	—

MissionJournalUI

MonoBehaviour

BLInformatique.RealFPS

AddComponentMenu: BLInformatique/Real FPS/Missions/UI/ Mission Journal UI



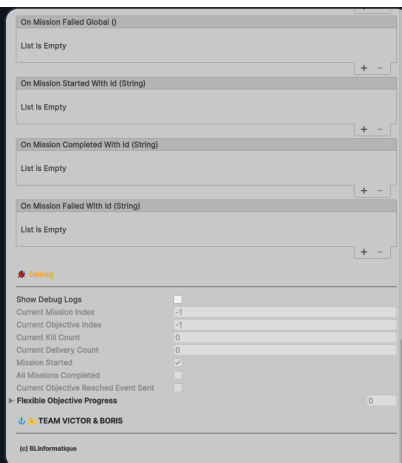
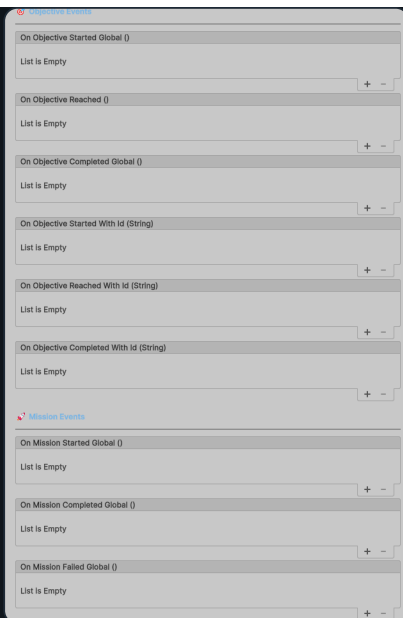
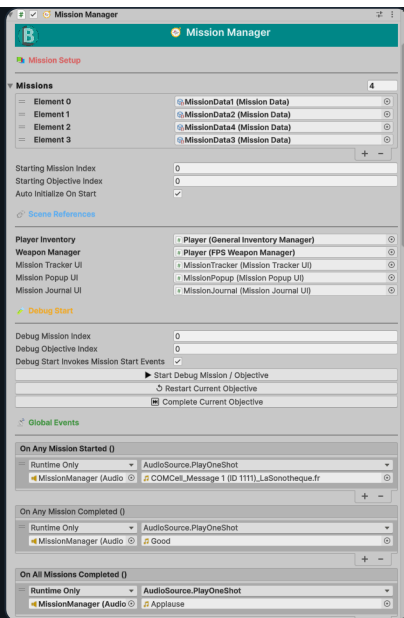
Field	Type	Tooltip	Hints
↓			
isOpen	Boolean	True if the journal is currently open.	Private Serialized
↓			
rootPanel	GameObject	Optional root panel used to show or hide the journal.	Object Reference
journalHeaderText	TMP_Text	Text field used for the journal header.	Object Reference
missionTitleText	TMP_Text	Text field used for the active mission title.	Object Reference

Field	Type	Tooltip	Hints
missionDescriptionText	TMP_Text	Text field used for the active mission description.	Object Reference
objectivesContainer	Transform	Parent transform where objective entry prefabs are instantiated.	Object Reference
objectiveEntryPrefab	MissionObjectiveEntryUI	Objective entry prefab used in the journal list.	Object Reference
↓			
useOldInput	Boolean	If true, the journal can be toggled with the old Input System.	—
toggleJournalKey	KeyCode	Keyboard key used to open or close the journal with the old Input System.	Enum
useNewInput	Boolean	If true, the journal can be toggled with the new Input System.	—
toggleJournalAction	InputAction	Input action used to open or close the journal with the new Input System.	—
↓			
missionManager	MissionManager	MissionManager used as the data source for this journal. If empty, one is searched automatically.	Object Reference
startHidden	Boolean	If true, the journal starts hidden on Awake.	—
journalHeaderLabel	String	Header label shown at the top of the journal.	—

▼ MissionManager MonoBehaviour

BLInformatique.RealFPS

AddComponentMenu: BLInformatique/Real FPS/Missions/ Mission Manager



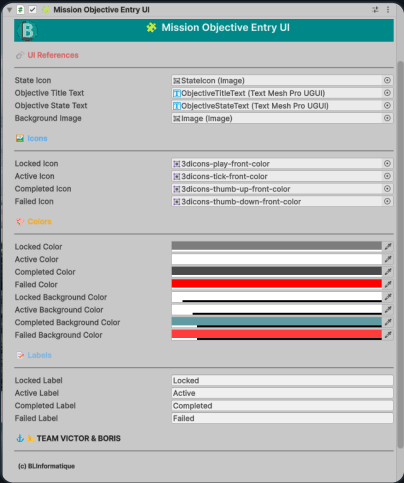
Field	Type	Tooltip	Hints
currentMissionIndex	Int32	Current mission index at runtime.	Private Serialized
currentObjectiveIndex	Int32	Current objective index at runtime.	Private Serialized
currentKillCount	Int32	Current kill progress for the active KillEnemy objective.	Private Serialized
currentDeliveryCount	Int32	Current cargo count inside the active delivery zone.	Private Serialized
missionStarted	Boolean	True if the current mission has started.	Private Serialized
allMissionsCompleted	Boolean	True if all configured missions are completed.	Private Serialized
currentObjectiveReachedEventSent	Boolean	True if the current objective reached event has already been sent.	Private Serialized
flexibleObjectiveProgress	List<FlexibleObjectiveProgress>	Progress remembered for future objectives in Flexible missions.	List / Array Private Serialized

Field	Type	Tooltip	Hints
↓			
missions	MissionData[]	Ordered list of missions played by this manager.	List / Array
startingMissionIndex	Int32	Index of the mission to start with.	—
startingObjectiveIndex	Int32	Index of the objective to start with when using Initialize or debug start.	—
autoInitializeOnStart	Boolean	If true, MissionManager starts automatically on Start.	—
↓			
playerInventory	GeneralInventoryManager	Player inventory used to validate OwnItem objectives.	Object Reference
weaponManager	FPSWeaponManager	Weapon manager used to validate EquipItem objectives.	Object Reference
missionTrackerUI	MissionTrackerUI	Optional tracker UI updated at runtime.	Object Reference
missionPopupUI	MissionPopupUI	Optional popup UI used for mission notifications.	Object Reference
missionJournalUI	MissionJournalUI	Optional journal UI updated at runtime.	Object Reference
↓			
debugMissionIndex	Int32	Mission index used by the inspector debug button.	—

Field	Type	Tooltip	Hints
debugObjectiveIndex	Int32	Objective index used by the inspector debug button.	—
debugStartInvokesMissionStartEvents	Boolean	If true, debug start methods also invoke mission started events and popups.	—
startDebugMissionObjectiveButton	Boolean	Starts the selected debug mission and objective.	—
restartCurrentObjectiveButton	Boolean	Restarts the current mission at the current objective index.	—
completeCurrentObjectiveButton	Boolean	Completes the current objective.	—
↓			
onAnyMissionStarted	UnityEvent	Invoked when any mission starts.	UnityEvent
onAnyMissionCompleted	UnityEvent	Invoked when any mission is completed.	UnityEvent
onAllMissionsCompleted	UnityEvent	Invoked when all missions are completed.	UnityEvent
↓			
onObjectiveStartedGlobal	UnityEvent	Invoked when the current objective starts.	UnityEvent
onObjectiveReached	UnityEvent	Invoked when the current objective conditions are reached.	UnityEvent
onObjectiveCompletedGlobal	UnityEvent	Invoked when the current objective is completed.	UnityEvent

Field	Type	Tooltip	Hints
onObjectiveStartedWithId	MissionStringEvent	Invoked when the current objective starts. Parameter = objectiveId.	UnityEvent
onObjectiveReachedWithId	MissionStringEvent	Invoked when the current objective conditions are reached. Parameter = objectiveId.	UnityEvent
onObjectiveCompletedWithId	MissionStringEvent	Invoked when the current objective is completed. Parameter = objectiveId.	UnityEvent
↓			
onMissionStartedGlobal	UnityEvent	Invoked when a mission starts.	UnityEvent
onMissionCompletedGlobal	UnityEvent	Invoked when a mission is completed.	UnityEvent
onMissionFailedGlobal	UnityEvent	Invoked when a mission fails.	UnityEvent
onMissionStartedWithId	MissionStringEvent	Invoked when a mission starts. Parameter = missionId.	UnityEvent
onMissionCompletedWithId	MissionStringEvent	Invoked when a mission is completed. Parameter = missionId.	UnityEvent
onMissionFailedWithId	MissionStringEvent	Invoked when a mission fails. Parameter = missionId.	UnityEvent
↓			
showDebugLogs	Boolean	If true, prints mission progression logs in the Console.	—

AddComponentMenu: BLInformatique/Real FPS/Missions/UI/  Mission Objective Entry UI



Field	Type	Tooltip	Hints
↓			
statelcon	Image	Optional icon image used to display the objective state.	Object Reference
objectiveTitleText	TMP_Text	Text field used for the objective title.	Object Reference
objectiveStateText	TMP_Text	Text field used for the objective state label.	Object Reference
backgroundImage	Image	Optional background image tinted according to state.	Object Reference
↓			
lockedIcon	Sprite	Icon used for locked objectives.	Object Reference
activeIcon	Sprite	Icon used for active objectives.	Object Reference
completedIcon	Sprite	Icon used for completed objectives.	Object Reference
failedIcon	Sprite	Icon used for failed objectives.	Object Reference
↓			
lockedColor	Color	Text and icon color used for locked objectives.	—
activeColor	Color	Text and icon color used for active objectives.	—
completedColor	Color	Text and icon color used for completed objectives.	—
failedColor	Color	Text and icon color used for failed objectives.	—

Field	Type	Tooltip	Hints
lockedBackgroundColor	Color	Optional background color used for locked objectives.	—
activeBackgroundColor	Color	Optional background color used for active objectives.	—
completedBackgroundColor	Color	Optional background color used for completed objectives.	—
failedBackgroundColor	Color	Optional background color used for failed objectives.	—
↓			
lockedLabel	String	Text label shown for locked objectives.	—
activeLabel	String	Text label shown for active objectives.	—
completedLabel	String	Text label shown for completed objectives.	—
failedLabel	String	Text label shown for failed objectives.	—

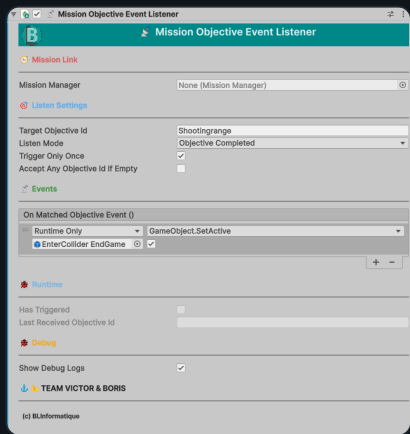
▼

MissionObjectiveEventListener

MonoBehaviour

BLInformatique.RealFPS

AddComponentMenu: BLInformatique/Real FPS/Missions/  Mission Objective Event Listener



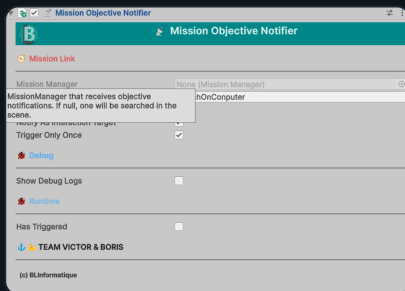
Field	Type	Tooltip	Hints
↓			
missionManager	MissionManager	MissionManager used as the event source. If empty, one will be searched automatically in the scene.	Object Reference
↓			
targetObjectveId	String	Objective identifier to listen for.	—

Field	Type	Tooltip	Hints
listenMode	MissionObjectiveListenMode	Mission objective event type to react to.	Enum
triggerOnlyOnce	Boolean	If true, the listener only reacts once.	—
acceptAnyObjectiveIdIfEmpty	Boolean	If true, an empty targetObjectiveId will match any objective.	—
↓			
onMatchedObjectiveEvent	UnityEvent	Invoked when the configured objective event is matched.	UnityEvent
↓			
hasTriggered	Boolean	True if this listener already reacted once.	—
lastReceivedObjectiveId	String	Last received objective identifier.	—
↓			
showDebugLogs	Boolean	If true, prints listener activity in the Console.	—

▼ 📦 MissionObjectiveNotifier MonoBehaviour

BLInformatique.RealFPS

AddComponentMenu: BLInformatique/Real FPS/Missions/👤 Mission Objective Notifier



Field	Type	Tooltip	Hints
missionManager	MissionManager	MissionManager that receives objective notifications. If null, one will be searched in the scene.	Object Reference
objectiveId	String	Objective identifier sent to the MissionManager.	—

Field	Type	Tooltip	Hints
notifyOnInteract	Boolean	If true, this component can notify the mission when interacted with by the player.	—
notifyAsInteractionTarget	Boolean	If true, this component sends its GameObject as interaction target.	—
triggerOnlyOnce	Boolean	If true, the notifier only sends the event once after successful validation.	—
↓			
interactionDisplayName	String	User-facing object name displayed by InteractionRaycaster. If empty, the GameObject name is used.	—
interactionVerb	String	User-facing action verb displayed by InteractionRaycaster.	—
↓			
showDebugLogs	Boolean	If true, prints debug logs in the Console.	—
↓			
hasTriggered	Boolean	True if this notifier already triggered successfully once.	—

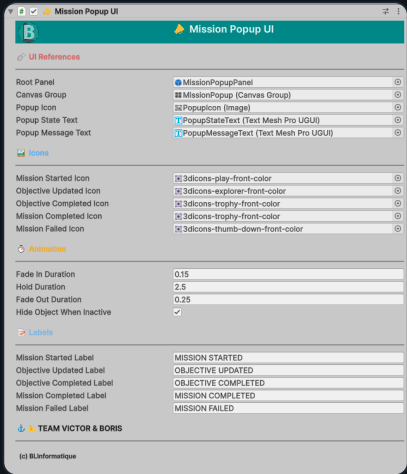
▼

MissionPopupUI

MonoBehaviour

BLInformatique.RealFPS

AddComponentMenu: BLInformatique/Real FPS/Missions/UI/🔥 Mission Popup UI



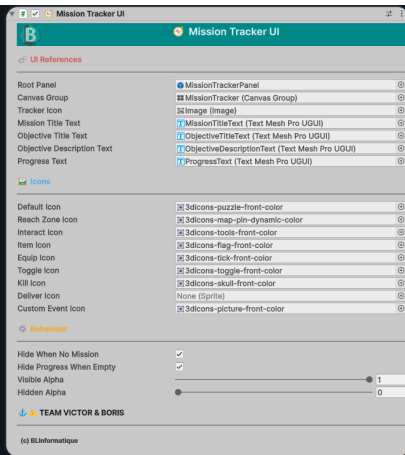
Field	Type	Tooltip	Hints
rootPanel	GameObject	Optional root panel used to show or hide the popup.	Object Reference

Field	Type	Tooltip	Hints
canvasGroup	CanvasGroup	CanvasGroup used for popup fade animation.	Object Reference
popupIcon	Image	Optional icon displayed in the popup.	Object Reference
popupStateText	TMP_Text	Text field used for the popup state label.	Object Reference
popupMessageText	TMP_Text	Text field used for the popup message.	Object Reference
↓			
missionStartedIcon	Sprite	Icon used for MissionStarted popups.	Object Reference
objectiveUpdatedIcon	Sprite	Icon used for ObjectiveUpdated popups.	Object Reference
objectiveCompletedIcon	Sprite	Icon used for ObjectiveCompleted popups.	Object Reference
missionCompletedIcon	Sprite	Icon used for MissionCompleted popups.	Object Reference
missionFailedIcon	Sprite	Icon used for MissionFailed popups.	Object Reference
↓			
fadeInDuration	Single	Duration of the popup fade in animation.	—
holdDuration	Single	Duration of the popup visible hold time.	—
fadeOutDuration	Single	Duration of the popup fade out animation.	—
hideObjectWhenInactive	Boolean	If true, the popup panel is deactivated when not in use.	—
↓			
missionStartedLabel	String	Label shown for mission started popups.	—
objectiveUpdatedLabel	String	Label shown for objective updated popups.	—
objectiveCompletedLabel	String	Label shown for objective completed popups.	—
missionCompletedLabel	String	Label shown for mission completed popups.	—
missionFailedLabel	String	Label shown for mission failed popups.	—

MissionTrackerUI
MonoBehaviour

BLInformatique.RealFPS

AddComponentMenu: BLInformatique/Real FPS/Missions/UI/  Mission Tracker UI



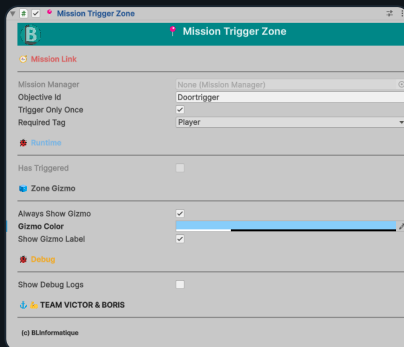
Field	Type	Tooltip	Hints
↓			
rootPanel	GameObject	Optional root panel used to show or hide the tracker.	Object Reference
canvasGroup	CanvasGroup	Optional CanvasGroup used for fade visibility.	Object Reference
trackerIcon	Image	Optional icon displayed next to the mission title.	Object Reference
missionTitleText	TMP_Text	Text field used for the mission title.	Object Reference
objectiveTitleText	TMP_Text	Text field used for the objective title.	Object Reference
objectiveDescriptionText	TMP_Text	Text field used for the objective description.	Object Reference
progressText	TMP_Text	Text field used for optional progress display.	Object Reference
↓			
defaultIcon	Sprite	Default icon used when no specific icon is assigned.	Object Reference
reachZoneIcon	Sprite	Icon used for ReachZone objectives.	Object Reference
interactIcon	Sprite	Icon used for InteractWithTarget objectives.	Object Reference
itemIcon	Sprite	Icon used for OwnItem and PickupItem objectives.	Object Reference
equipIcon	Sprite	Icon used for EquipItem objectives.	Object Reference
toggleIcon	Sprite	Icon used for ToggleState objectives.	Object Reference
killIcon	Sprite	Icon used for KillEnemy objectives.	Object Reference

Field	Type	Tooltip	Hints
deliverIcon	Sprite	Icon used for DeliverCargo objectives.	Object Reference
customEventIcon	Sprite	Icon used for CustomEvent objectives.	Object Reference
↓			
hideWhenNoMission	Boolean	If true, the tracker is hidden when no mission is active.	—
hideProgressWhenEmpty	Boolean	If true, empty progress text is hidden automatically.	—
visibleAlpha	Single	Alpha applied when the tracker is visible. • Range [0..1]	Range
hiddenAlpha	Single	Alpha applied when the tracker is hidden. • Range [0..1]	Range

MissionTriggerZone MonoBehaviour

BLInformatique.RealFPS

AddComponentMenu: BLInformatique/Real FPS/Missions/  Mission Trigger Zone • Requires: Collider



Field	Type	Tooltip	Hints
↓			
missionManager	MissionManager	MissionManager receiving the trigger event.	Object Reference
objectiveId	String	Objective ID sent to the MissionManager.	—
triggerOnlyOnce	Boolean	If true, the zone triggers only once after successful validation.	—
requiredTag	String	Required tag on the entering object.	—
↓			
hasTriggered	Boolean	True if this trigger has already been accepted by the MissionManager.	—

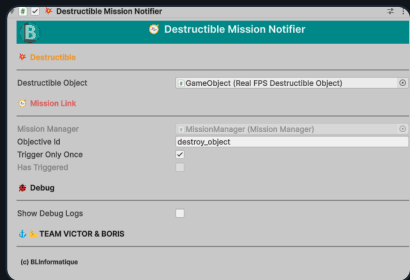
Field	Type	Tooltip	Hints
↓			
alwaysShowGizmo	Boolean	If true, the trigger gizmo is always visible in the Scene view.	—
gizmoColor	Color	Color of the trigger gizmo.	—
showGizmoLabel	Boolean	If true, a label is displayed in the Scene view for this zone (Editor only).	—
↓			
showDebugLogs	Boolean	If true, prints debug logs in the Console.	—

▼

DestructibleMissionNotifier
MonoBehaviour

BLInformatique.RealFPS.Missions

AddComponentMenu: BLInformatique/Real FPS Mission System/★ Destructible Mission Notifier • Requires: RealFPSDestructibleObject



Field	Type	Tooltip	Hints
↓			
destructibleObject	RealFPSDestructibleObject	Destructible object that triggers this mission notification.	Object Reference
↓			
missionManager	MissionManager	Mission manager receiving the objective notification.	Object Reference
objectiveId	String	Objective identifier sent to the MissionManager when the object is destroyed.	—
triggerOnlyOnce	Boolean	If enabled, the notifier can only complete the objective once.	—
hasTriggered	Boolean	True after the MissionManager accepts the objective notification.	—
↓			

Field	Type	Tooltip	Hints
showDebugLogs	Boolean	Display mission notification messages in the Console.	—

v1.0 • BLInformatique • realfps@blinformatique.fr