

Welcome to Balancy Documentation

Balancy is a superhero that swoops in to save game studios from the clutches of mediocrity. Our mission? To increase LTV and help studios triumph over competitors by customizing and personalizing gaming experiences.

What is Balancy?

Balancy is a robust, flexible tool designed to run Live Operations and manage in-game economies. It facilitates essential features such as virtual currencies, inventory systems, player progression, and more. Our core feature helps game developers run and analyze Live Operations, which, as a result, increases a game's LTV. LiveOps managers and Game Designers get convenient tools to work remotely with Game Events, Personalised Offers, A/B Testing, and Segmentation. Balancy requires minimum effort from engineers and helps them with many automated tools.

Why use Balancy?

Balancy provides extensive features to support your game's LiveOps and economy, whether you're developing a small indie game or a large-scale multiplayer project. It offers flexibility and customization to suit your specific needs. By using Balancy, you can save development time and focus more on gameplay and user experience.

Watch Video



[Join Our Discord](#)

How to use this guide?

This guide is divided into several sections:

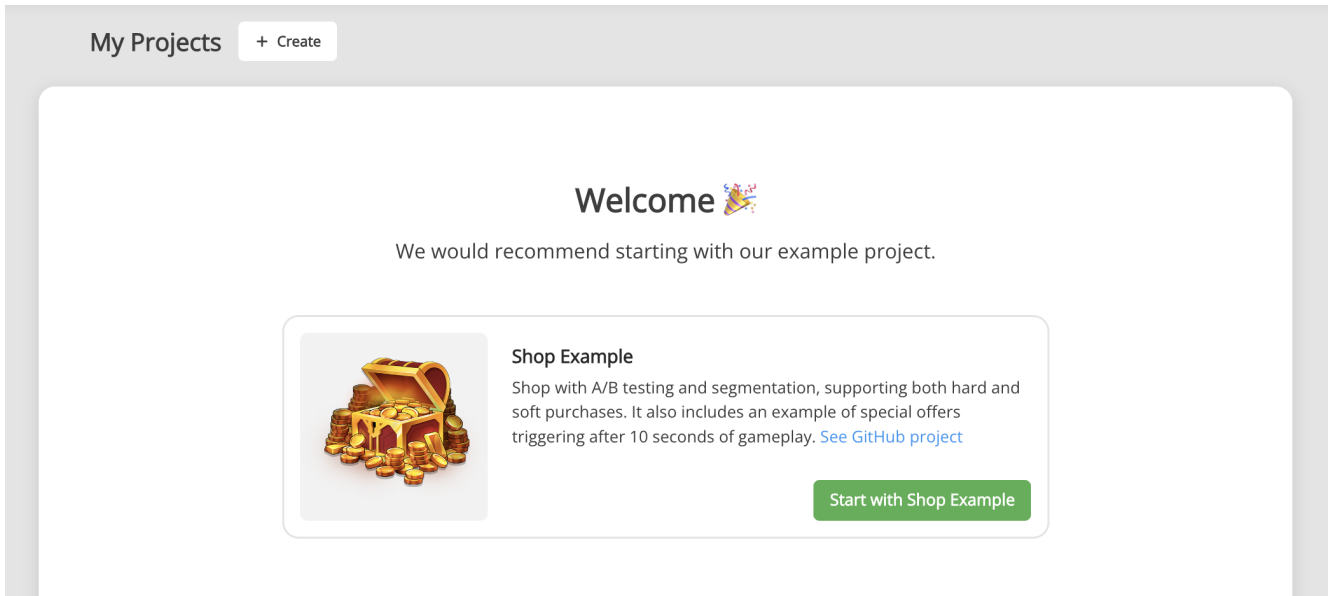
- **Quick Start:** This section covers setting up Balancy and running LiveOps.
- **LiveOps:** Here, you'll find more detailed instructions on using Live Operations.
- **Content Management System:** This section overviews the Balancy balance editor, inventory, and in-game economy.
- **Deploy:** Learn how to deliver data from Balancy to players.

Feel free to navigate through the documentation in a way that suits best to your needs. If you have any questions or issues, please don't hesitate to contact our support team. We're here to help!

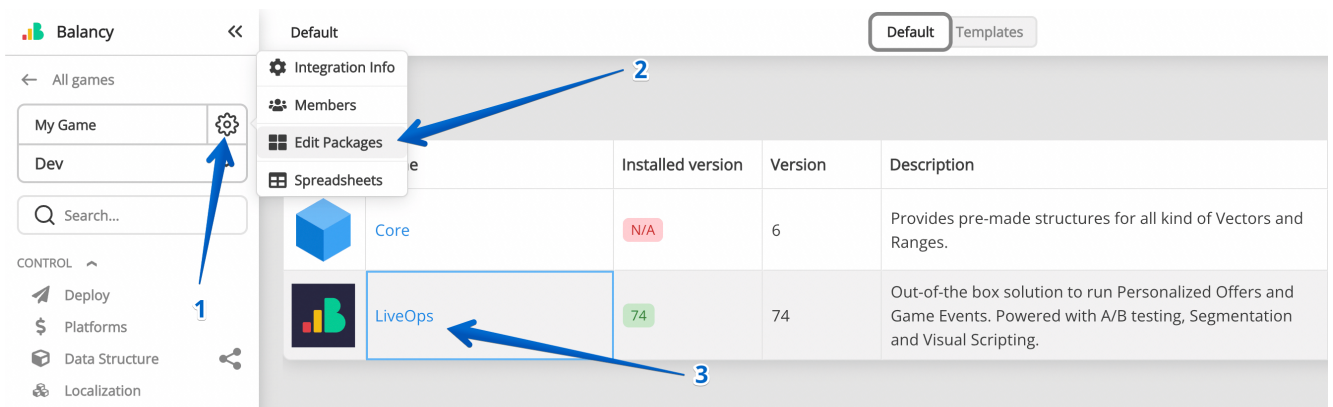
Step-by-step Integration

Step 1. Basic Integration (Platform) [30 minutes]

1. Create an account at [Balancy](#).
2. Once you see the dashboard, create a new game or use our [Template](#) (Recommended).



3. If you created a new game, install [LiveOps](#) package.



4. (Optional) Add team members to the project. It is possible to give personal read/write permissions to members.

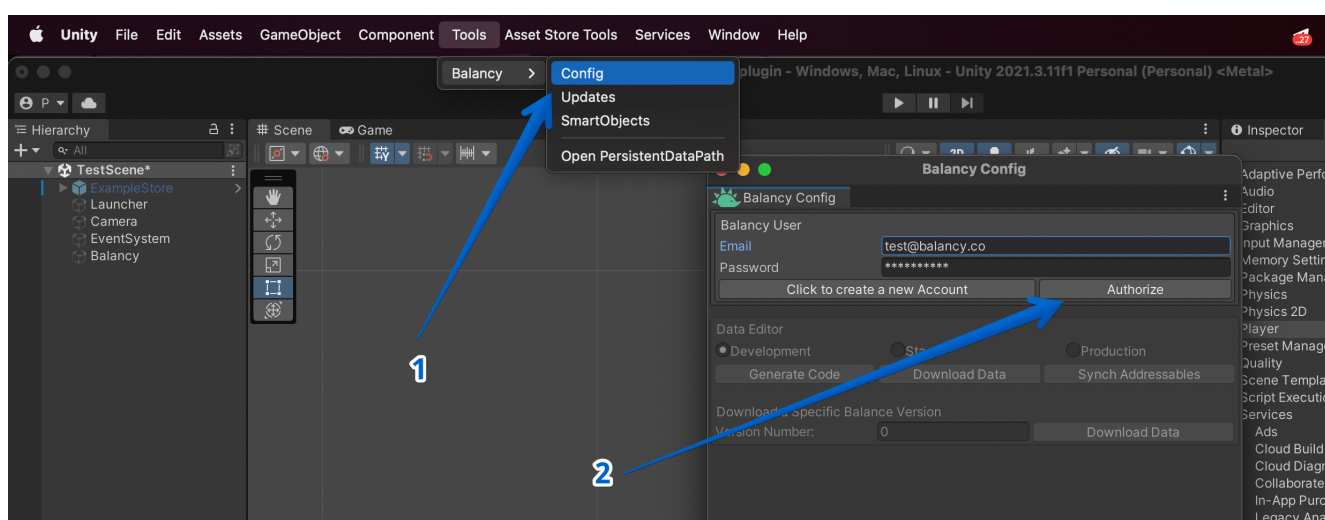
Integration Guide

Efficiently integrate Balancy into your gaming environment with our Unity plugin and initialize it with ease.

Plugin Installation

Ensure a smooth integration with these straightforward steps:

1. **Download Plugin:** Obtain the most recent version of the [plugin here](#).
2. **Import to Unity:** Incorporate the Balancy plugin into your Unity project.
3. **Authorization:**
 - Navigate to **Tools ► Balancy ► Config** post-compilation.
 - Use the Balancy platform credentials to authorize.



4. **Game Code Generation:**
 - Choose the relevant game.
 - Click **Generate Code** to proceed.

Launch Check List

Follow this comprehensive workflow for a seamless release of game update on App Store/Google Play:

Preparation Steps

1. Begin by [deploying](#) all recent changes to the **Dev** environment. Ensure to specify the minimum app version compatible with the new data.
2. Rigorously test the app within the **Dev** environment to confirm everything operates as intended.
3. Safeguard your progress by [migrating](#) data from **Dev** to **Stage** environment. This strategy maintains a stable version on **Stage** in the event of issues during the review phase, leaving **Dev** free for ongoing developments.

Build Process

1. Utilize Unity to [Download Data](#) from either **Dev** or **Stage** environments. At this point, both environments should align in terms of data.
2. **Critical Step:** [Build](#) your game specifying the **Production** environment in the **Init** method. Although the Production cloud may lack data, the previously injected information will temporarily suffice.
3. Thoroughly test the build. Address any issues, potentially looping back to step (1) for major fixes. Despite targeting the **Production** environment, the embedded higher-version data prevails.

Submission

1. Submit the polished build to App Store/Google Play.
2. Patiently await approval before releasing the build.

Post-Launch

1. Post-approval, execute a final [migration](#) of data from **Stage** to **Production**, mindful of the minimum version prerequisites.
2. With everything in place, you are now free to modify data within the **Production** environment, deploying real-time updates directly to your players.

Adhering to this checklist ensures a streamlined, efficient, and less error-prone release process, contributing to a smoother experience for your players and maintaining the integrity of your live product.

Live Operations

You've heard about Live Operations many times and may think you know everything about it. Yet, when you try to define the term it turns out to be vague and includes everything from in-game events and promotional sales to support and community management.

LiveOps strategies help games become living and evolving organisms versus the 'create – release – start a new project' approach. When you run a game as a service, the release date is the beginning of a new life. Like a new organism, your game is evolving, keeping players constantly coming back for more fun and unique experiences.

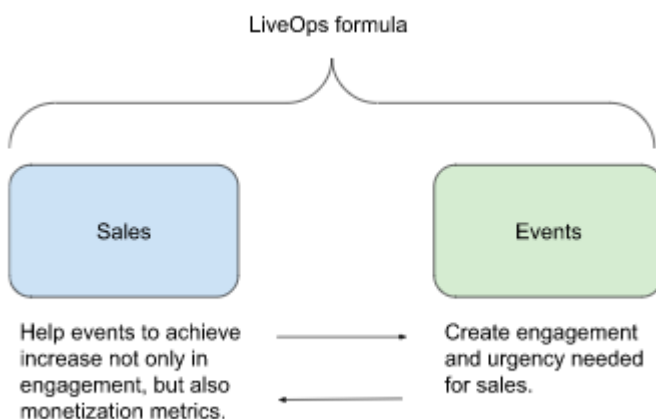
Let's define LiveOps the following way: they are changes and updates you bring into the game after the release date to engage your players, create long-term relationships with them, and increase LTV of your game.

Here are some examples of what LiveOps can be:

- Adding a skimming pricing(*) strategy to your shop
- Adjusting the frequency of ads to various segments of players
- Decorating a game for St. Valentine's Day and adding new festive skins
- Promoting your in-game tournament on social media, where your community gathers
- Bug fixes and app productivity optimization

(*) Skimming pricing initially sets a relatively high price for an offer and subsequently lowers it over time.

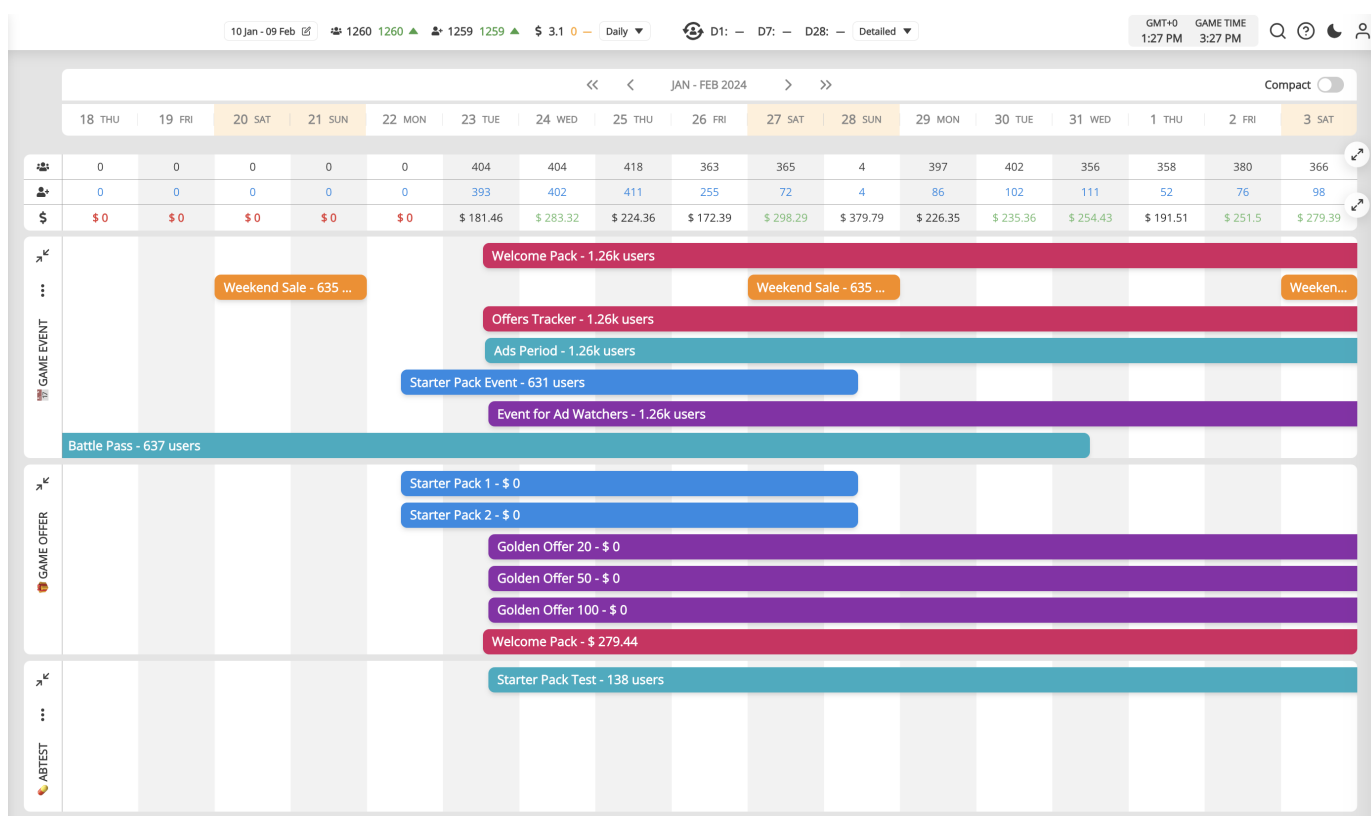
The most common and proven combination for your LiveOps strategy is combining in-game events and promotional sales. Sales are usually coupled with Events to create sense of urgency to buy. You can make a time-limited Black Friday sale with huge discounts or start selling new winter skins around Christmas as a part of your winter in-game season.



Dashboard

Welcome to the heart of your LiveOps strategy - the Dashboard! This is where you can get a high-level view of your game's activities and performance, all in one place. The dashboard is designed with game developers in mind, providing you with actionable insights and easy access to key tools for planning and managing in-game events, offers, A/B tests, push notifications, and more.

Overview



Your Dashboard is the control center for your game's LiveOps. At a glance, you can:

- View ongoing and scheduled in-game activities.
- Monitor the performance of your campaigns.
- React quickly to changes or issues.

Offers & Items

Item

An Item Document represents one entity within your game. It can be a sword, shield, wood, stone - anything from your game.

Parameter	Description
Name	The name of the Item
MaxStack	The maximum amount of Item, which can be put in one Slot

We want to keep this structure as universal as possible. Thus, we only added a few parameters. However, your game needs much more parameters than we provide. Just create a new Template inherited from **Item** and add as many parameters as you need. We have a great template package [Merge Game](#), which can teach you how to do that and how to operate with complicated item systems.

Store Items

Store Items section contains everything you can sell directly through the in-game [Shop](#) or via Game Offers.

ID	Sprite	Name	Price	Reward	Updated
259	StoreIcon_Chest_3	Epic Chest	[Gems x 3000]	[Chest x 1]	6 May, 2023 3:17 PM
262	StoreIcon_Gem_0	Tiny of Gems	Gems Pack 0 (\$1.99)	[Gems x 30]	6 May, 2023 3:17 PM
266	StoreIcon_Gem_1	Fistful of Gems	Gems Pack 1 (\$4.99)	[Gems x 80]	6 May, 2023 3:17 PM
270	StoreIcon_Gem_2	Pouch of Gems	Gems Pack 2 (\$9.99)	[Gems x 170]	6 May, 2023 3:17 PM
282	StoreIcon_Gem_3	Box of Gems	Gems Pack 3 (\$19.99)	[Gems x 360]	6 May, 2023 3:17 PM
304	StoreIcon_Gem_4	Chest of Gems	Gems Pack 4 (\$49.99)	[Gems x 950]	6 May, 2023 3:17 PM
313	StoreIcon_Gem_5	Vault of Gems	Gems Pack 5 (\$99.99)	[Gems x 2000]	6 May, 2023 3:17 PM
658	StoreIcon_Gold_0	Tiny of Gold	Gold Pack 0 (\$1.99)	[Gold x 150]	6 May, 2023 3:17 PM

Inventory

Inventory in games is a crucial feature for managing player-owned items and currencies. It tracks player acquisitions, whether through in-game purchases, rewards, or other means. Balancy seamlessly integrates inventory management, ensuring a smooth player experience.

Uses of Inventory

Inventory systems can be adapted for various game types, including:

- **In-Game Currencies:** Managing virtual money, gems, or resources.
- **Storage in Farm Games:** Keeping track of harvested crops, tools, and equipment.



- **Bag and Inventory in RPGs:** Holding weapons, potions, and other RPG essentials.

Purchases

Balancy facilitates a variety of purchase types to suit different game monetization strategies. Players can acquire:

1. [StoreItem](#)
2. [GameOffer](#)
3. [OfferGroup](#)

Explore the four methods to process purchases:

Direct Purchase

Balancy automates price checks and validations, informing the client about the outcome through a callback.

[Learn more about the purchase flow.](#)

Setup

- Install the Balancy Payments package via **Tools ► Balancy ► Updates**.
- Configure [Payments](#) in Balancy, providing necessary validation data.

Methods to initiate a purchase:

```
Balancy.LiveOps.Store.PurchaseStoreItem(StoreItem storeItem,
                                         System.Action<PurchaseProductResponseData> callback);
Balancy.LiveOps.GameOffers.PurchaseOffer(OfferInfo offerInfo,
                                         System.Action<PurchaseProductResponseData> callback);
Balancy.LiveOps.GameOffers.PurchaseOffer(OfferGroupInfo offerInfo,
                                         StoreItem storeItem,
                                         System.Action<PurchaseProductResponseData> callback);
```

Post-Purchase Notification

If you manage the transactions by yourself, use these methods to notify Balancy post-purchase, utilizing receipt validation. [Detailed purchase flow.](#)

Setup

- Ensure the Balancy Payments package is installed. **Tools ► Balancy ► Updates**.
- Complete [Payments](#) configuration for validation requirements.

Game Events

A Game Event is the first step for launching Game Offers. However, Game Events have a much broader meaning. You can use them to plan your in-game activities and events like Halloween, Tournaments, Weekends, and more. Adding Game Events increase your game's retention and engagement.

After installing the LiveOps package, you get a ready-to-use structure for Game Events, and the table looks like this:

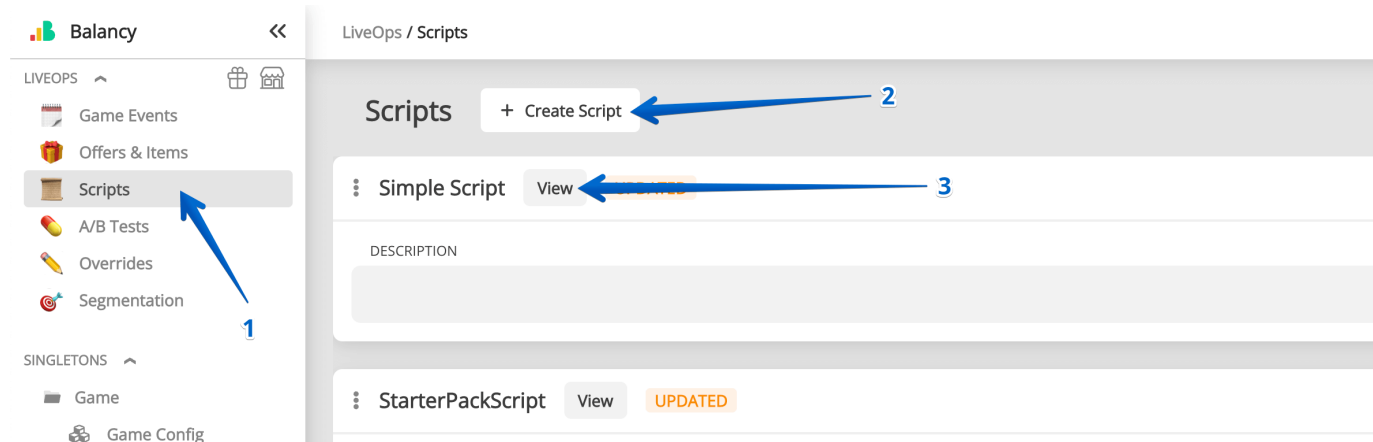
The screenshot displays two event configuration cards. The first card, titled 'Welcome Pack', has a 'View' button and a 'None' condition. The second card, titled 'Weekend Sale', has a 'View' button and a condition set to '&& [Days: Sunday, Saturday]'. Both cards include an 'ATTACHMENT' field, a 'COMMENTS' field, and a 'TAGS' field with an 'Add tag...' input.

Parameter	Description
Name	This name is used mostly for your convenience
Attachment	Attachment can be either Script, or Game Offer. The script is executed once the event starts. The script can have any logic, but generally, it's used to Segment players and give them Personalised Offers. You can read more about scripts in Visual Scripting section. The same way a Game Offer is activated when event starts, but it simply shows an offer without complex logic. You can learn more about Game Offers in Offers & Items section.
Condition	A condition required to meet to start the event. Leave it blank if you want to run this event all the time - this approach works for Starter Pack Offers. But if you have seasonal, weekend, or game anniversary events - you should specify the conditions. In most cases, the conditions are time-based.
FinishType	Defines how the Game Event should end. By default the Game Event deactivates once the Condition is false, you can change the type to Duration and specify how long the event lasts after activate. Keep in mind, after deactivation, if Condition is true, the Game Event starts again.

Visual Scripting

Visual Scripting is a tool designed to lower the entry barrier to programming. As code is more visible, it needs less abstract thinking to be understood. Any game designer or analyst can look at it and quickly understand the logic flow.

We've added VS to Balancy, so you can segment and limit your Game Offers to ensure that the correct player gets the right offer. The first version of VS is limited, but we plan to expand it so you can use it for anything, even game logic. The best part of VS is that it lets your designers manipulate game logic remotely and without distracting your engineering team.



Parameter	Description
Name	This name is used mainly for your convenience
Description	Helps other team members quickly understand the purpose of the Script

When you click on a **View** button of any Script, the Visual Scripting tool will launch and open the selected Script.

Scripts

In Visual Scripting, a script is similar to any scripts in other languages: it is a sequence of instructions or commands that are visually constructed using [Nodes](#) and [Links](#). These commands are then executed during runtime on the end user's device. Scripts are typically used to determine which Game Offers should be shown to an end user.

Creating a Script

Scripts are created in the Scripts page, which can be found under the [LiveOps](#) section of the Navigation panel. Upon creating a Script, it can be opened by pressing the [View](#) button next to the Script name, which will bring you to the Script View.

Using a Script

Activation By Game Event

After creating a Script, it can be activated in-game by first attaching it to a Game Event. The Script activates as soon as the Game Event itself activates when its conditions are met.

Note: If a Game Event has no conditions, it will immediately activate upon starting the game.

If the Script's Input Port is named exactly 'GameEvent' and is of Document type, it will automatically receive Game Event document which has activated this Script. It is useful to get access to Game Event's parameters.

Activation From Code

Documents can have parameters of Script type. Check Dynamic Reward script param of [Store Item](#) for example. If you create parameters of the Script type in your custom documents, you can execute scripts from the code.

```
gameEvent.Script.Run( inputs:new Dictionary<string, object> { { "Number", 5 } }, onExit:(exit:string, outputs:Dictionary<string,object> ) =>
{
    if (outputs != null)
    {
        if (outputs.TryGetValue("Output", out var output:object ))
            Debug.LogError(message:$"output = {output}");
    }
});
```

On Balancy side the script for this example would have been like this:

Toolbar

In Visual Scripting, the toolbar contains useful tools to help you while building your own [Scripts](#). Upon opening a Script, it can be found at the bottom of the Script view.

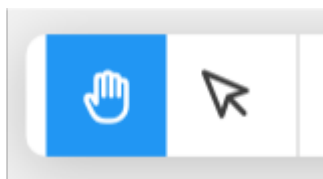
Tools

Here are the tools you can use within the toolbar. Their descriptions contain both the tool icon and its activation hotkey.

Grab Mode (G)

Switches to Grab Mode. This mode allows you to pan around the Script. It is possible to select and interact with [Nodes](#), [Ports](#), and [Links](#) in this mode, but prevents you from dragging to select multiple nodes.

You will be able to tell if you're in Grab Mode if the Grab icon is highlighted and the Select icon is not.



Note: While in Grab Mode, you can quickly switch to [Select Mode](#) by holding the `Shift` button.

Select Mode (V)

Switches to Select Mode. This mode allows you to select multiple nodes; simply hold and drag the cursor over the nodes you wish to select. It is possible to select and interact with [Nodes](#), [Ports](#), and [Links](#) in this mode, but prevents you from panning around the Script.

You will be able to tell if you're in Select Mode if the Select icon is highlighted and the Grab icon is not.



Undo (`Ctrl + Z`) and Redo (`Ctrl + Y`)

The Undo tool undoes an action taken within a Script. Conversely, the Redo tool redoes a previously undone action. These tools are helpful if you ever want to quickly revert a mistake you made, or if you simply want to compare different values quickly.

Nodes

In Visual Scripting, nodes are fundamental building blocks that represent specific operations or actions in a Script. Here are some examples of actions that a node can do:

- Some nodes take in some inputs, processes them, then outputs some values (check out [Operator Node](#))
- Some nodes simply redirect the [Flow](#) of the Script depending on a condition (check out [Branch Node](#))
- Some nodes do some system actions (check out [Log Node](#))

Nodes come in different sizes and shapes, are made up of different kinds of [Ports](#), and can be linked to other nodes through [Links](#), but they are all essential parts of a Script in order for that Script to do something.

Creating Nodes

To create a node, simply right-click on the Script View. This will open up the Node Context Menu, where you can search for and choose the node you'd like to add. When hovered, each node in the menu will also show a short description about what they do. Once you found the node you'd like to add, simply click on the menu option, and the node should be added on the Script View near where you right-clicked.

Ports

In Visual Scripting, a port is found on a node and functions as a point of connection that other nodes can connect into. Ports cannot exist by themselves, and are always attached to nodes. Ports can be connected to other ports using [Links](#). Once connected, a port can only receive values from, or be [flowed into](#), its connecting port.

Note: There are rules about connecting ports to each other. Some ports can only connect to one other port, while some can connect to many ports. Check out the [Link Rules](#) for more details.

In Ports and Out Ports

There are two kinds of ports:

- In Ports - ports going into a node
- Out Ports - ports going out of a node

It is easy to tell them apart by checking which side of the node the ports are in. In Ports are always found on the left side of a node, while Out Ports are always found on the right side of a node. This also holds true for the Script itself; the Script's In Ports are always on the left side of the Script View, while the Script's Out Ports are always on the right side of the Script View.

The In Ports include [Enter Ports](#) and [Input Ports](#), while the Out Ports include [Exit Ports](#) and [Output Ports](#).

Note: One rule about connecting ports to each other is that, generally, In Ports cannot be connected to other In Ports. Similarly, Out Ports cannot be connected to other Out Ports. However, there is an exception. Check out the [Link Rules](#) for more details.

Links

In Visual Scripting, a link is the connection between ports and allows nodes to talk to each other. Similar to [Ports](#), links cannot exist by themselves, and have to always be connected to a port on both ends. Depending on whether a link connects [Flow Ports](#) or [Value Ports](#) together, a link may carry either a [flow signal](#) or a value between ports.

Note: Check out the [Flow: Advanced](#) section to learn more.

Link Rules

Outlined below are all the rules about creating a link between ports.

1. An In Port cannot be connected to itself or other In Ports, and an Out Port cannot be connected to itself or other Out Ports.

This is the most basic rule that concerns [In Ports and Out Ports](#), the most general categories of ports. This rule is added to keep the [Flow](#) of the Script in one direction.

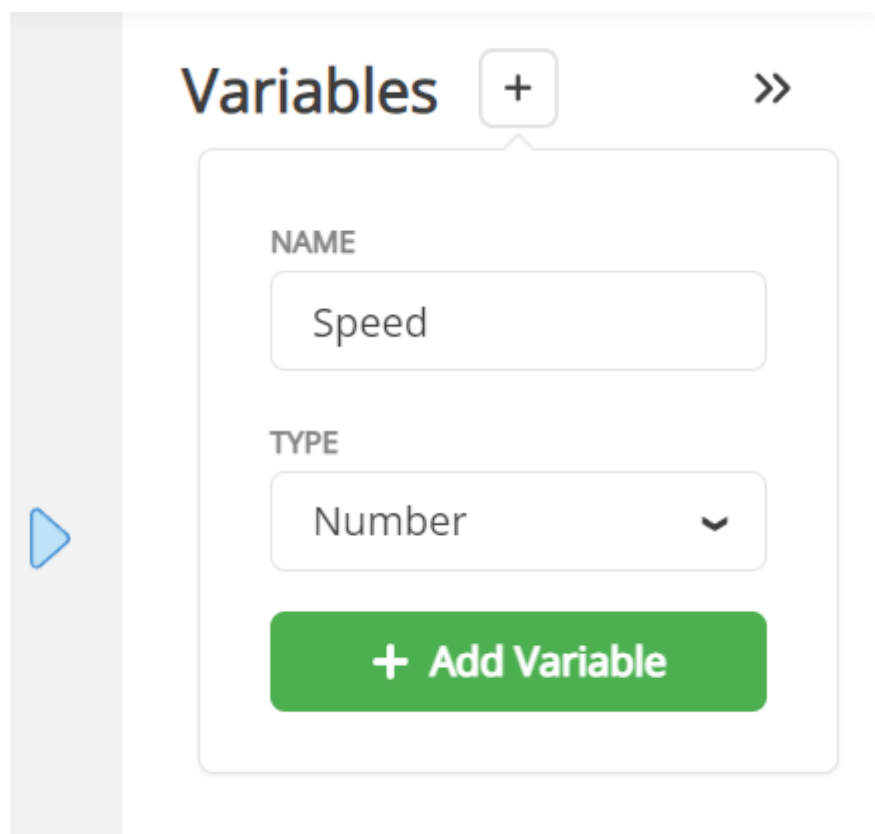
However, there is an exception. The In Ports of a Script -- found on the left side of the Script View -- can be connected to the In Ports of a node. Similarly, the Out Ports of a Script -- found on the right side of the Script View -- can be connected to the Out Ports of a node. This exception is done solely for ease of use. Although, this rule still applies the same way for Script's In Ports and Out Ports.

Variables

In Visual Scripting, a variable is used to store values into and read values from during the lifetime of a [Script](#). The value is stored temporarily until the Script finishes running, and cannot be accessed from outside the Script. See the [Use Cases](#) and [Limitations](#) of variables to know more about what variables can and cannot do.

Creating Variables

Most interactions with variables are done in the Variables Panel on the right side of the Script View. A variable can be added by first clicking on the plus icon at the top of the Variables Panel, inputting a Name and variable Type, and finally clicking on the green **+ Add Variable** button. Once a variable is added, depending on its type, a default value can be set to it.



Using Variables

Once a variable is created, it can be interacted with in the Script through these nodes:

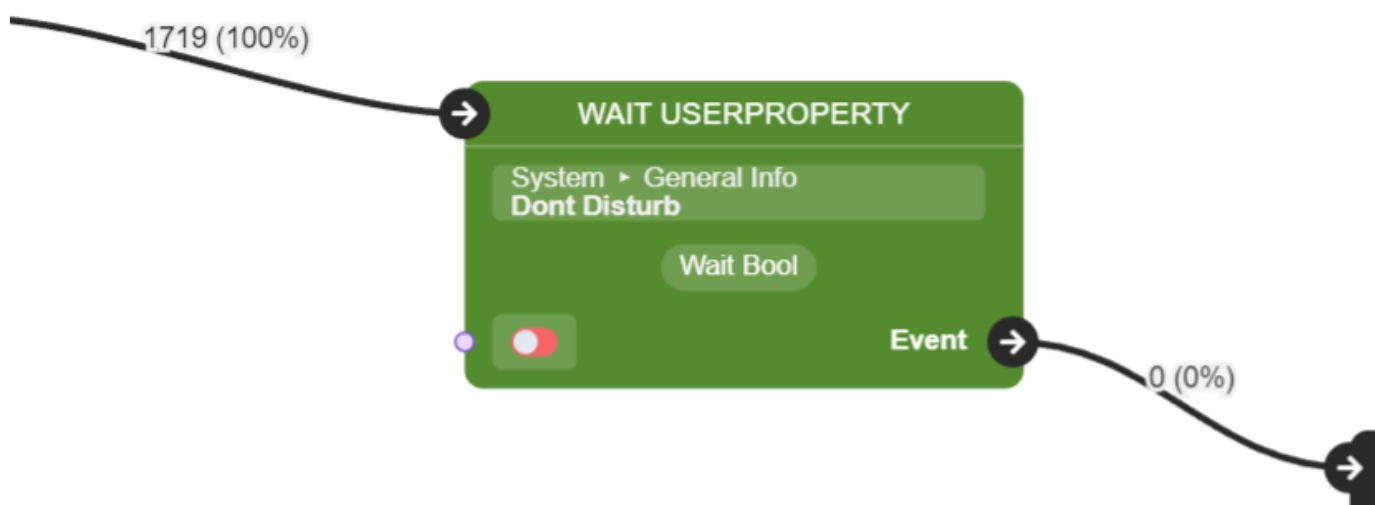
- **Get Variable Node** - Gets the value stored in a variable and outputs it through its `Value` Output Port. The type of the Output Port depends on the type of the variable.

Analytics

In Visual Scripting, the analytics currently available are mainly to provide more information about your Game Offers and how they are performing. There are two kinds of analytics shown: User Counts and Game Offer Revenue from Purchases.

User Counts

The User Counts analytics can be seen on top of the [links](#) between [Flow Ports](#). These counts show how many end users have gone through which nodes and taken which paths through your Scripts. The percentage next to the user count on a path is the percentage of end users that got through this path compared to the end users that entered this Script. These can be helpful when determining how many end users have purchased or declined an Offer.



Warning: Currently, if you have any [link loops](#) in your Script, the user count on the paths in these loops may exceed the number of end users entering the Script, causing them to have percentages greater than 100%. This is a known issue; we'll update this warning once this issue has been fixed.

Game Offer Revenue from Purchases

The Revenue from Purchases are only available and tracked for Activate Offer nodes. This analytics can be found next to the node's Purchased exit port. Once your Game Offers are being purchased and getting some revenue, the revenue will be shown here in USD.

All Nodes

Dynamic Nodes

Nodes in this category can change their ports depending on the option that you pick from the node's selection menu.

Operator

The Operator Node takes in the values from its Input Ports, **A** and **B**, processes them depending on the chosen operator, and outputs the resulting value into the **Result** Output Port. The operations that can be chosen are listed below.

Numbers

ADD

Returns the sum of the inputs **A** and **B**.

Example:

A = 5; **B** = 10

Result = 15



SUBTRACT

Returns the difference between the inputs **A** and **B**.

Example:

A = 5; **B** = 10

Result = -5

In-Game Store

The In-Game Store is the place to generate money for your game. A good store usually has a variety of items players need during the game. For the convenience of navigation, developers usually split all the content into different pages, such as Gems, Gold, Limited Offers, Super Packs, Event Items, etc. Each page contains several slots with different prices.

Follow the instructions to add a **Store** to your game:

1. Add the **LiveOps** package to your game.
2. Open **In-Game Store** section from the navigation panel.
3. **+ Create New Game Store** with as many pages and slots as you need.

The screenshot displays the 'In-Game Store' management interface. On the left is a navigation sidebar with sections like 'CONTROL', 'LIVEOPS', 'SINGLETONS', and 'DOCUMENTS'. The main area shows a grid of six 'Gems' items, each with a price and an analytics table. The items are labeled 'Gems 1' through 'Gems 6' and contain 10, 50, 100, 250, 500, and 1000 gems respectively. Each item has a price tag and a corresponding analytics table with columns for Total Revenue, Revenue Last Month, Revenue Last 7 Days, Total Purchases, Purchases Last Month, and Purchases Last 7 Days.

Item	Price	Total Revenue	Revenue Last Month	Revenue Last 7 Days	Total Purchases	Purchases Last Month	Purchases Last 7 Days
Gems 1	\$1.99	\$1.23k	\$131.34	\$25.87	619 (398)	66 (42)	13 (11)
Gems 2	\$8.99	\$5.67k	\$503.44	\$53.94	631 (425)	56 (28)	6 (5)
Gems 3	\$17.99	\$10.99k	\$1.42k	\$287.84	611 (387)	79 (47)	16 (13)
Gems 4	\$34.99	\$20.12k	\$1.68k	\$349.9	575 (402)	48 (35)	10 (9)
Gems 5	\$59.99	\$34.55k	\$3.24k	\$539.91	576 (395)	54 (34)	9 (9)
Gems 6	\$99.99	\$53.29k	\$5.9k	\$499.95	533 (349)	59 (37)	5 (4)

You can preview your work using our [Demo](#) project. It should look like this:

A/B Tests

A/B testing allows developers to run a controlled experiment between two or more versions of something in their game to determine which one is more effective. Your audience is segmented into the control group (current performance – no changes) and your test group.

A/B testing is a great way to find the best pricing and game difficulty or test any of your hypotheses.



Before using, all A/B Tests must be added to the table.

Parameter	Description
Name	The name of the Test.
Groups	Each user gets one random group.
Target Audience	The portion of your total users participating in the test.
Users Type	Defines the kind of users to target the test: New , Old , or All . If the app is opened for the first time, the user is considered a new one until he restarts the app.
Concurrent Test	Such test ignore the fact that some other tests can be active. All concurrent test will be activated for your users and won't affect Non-concurrent tests.
Conditions	Conditions that are required to start the A/B Test. After A/B Test starts for a player, it won't be stopped for him even if the conditions turn to False.
PreInitLaunch	Defines if the A/B Test can be launched during PreInit and before Balancy OnReadyCallback is executed. Read more here

Segmentation

User segmentation is the process of separating users into distinct groups, or segments, based on shared characteristics. Developers might segment users based on language preferences, product version, geographical region, or user persona. All that is needed is to define segment's **Condition** using one or several [User Properties](#).

Balancy has several built-in segments:

Monetary

Defines how much in total the player has already paid in the game:

- Killer Whale $\geq \$1500$
- Whale $\geq \$500$
- Orca $\geq \$150$
- Dolphin $\geq \$30$
- Minnow $\geq \$1$
- None $\$0$

Frequency

Defines how often the player makes purchases.

- F0 (None)
- F1 (Daily)
- F3 (2-4 a Week)
- F7 (Weekly)
- F14 (Once per 2 Weeks)
- F30 (Monthly)
- F60 (Rarely)

Recency

Defines how much time passed since the last purchase.

- None (never)
- R0 (<1 day)
- R1 (1-2 days)
- R2 (2-7) days
- R7 (7+ days)

MaxPay

Defines the maximum one-time payment the player can afford.

User Properties

User Properties allow you to store specific parameters for each player, aiding in audience [segmentation](#), triggering [Game Events](#), and customizing game logic with [Scripts](#).

Setting Up User Properties

1. For a new [Condition](#), opt for **User Property** ► **Primitive** or **User Property** ► **In Range**:

The screenshot shows the 'Segmentation' interface with a table of existing segments and a 'CONDITIONS' panel. A dropdown menu is open, showing options for 'USER PROPERTY' including 'In Range', 'Primitive', and 'Empty'. A blue arrow points to the 'Primitive' option.

NAME & ADDITIONAL INFO	CONDITIONS
My Segment	[Select item...] x
Frequency/F0 (None) How Often player makes purchases	[User Prop: ...] Purchase <= 0
Frequency/F1 (Daily) How Often player makes purchases	[User Prop: ...] Purchase ∈ [0 , 2)
Frequency/F3 (2-4 a Week) How Often player makes purchases	[User Prop: ...] Purchase ∈ [2 , 6)

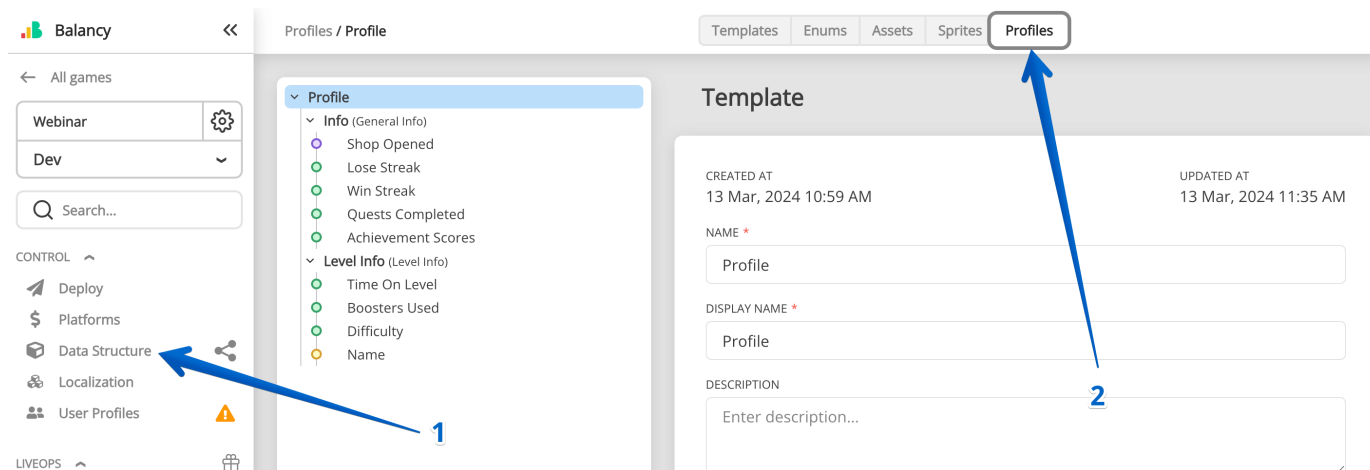
2. Pick a property, like **Level**, for example:

The screenshot shows the 'Segmentation' interface with the same table of segments. The dropdown menu is now open, showing a list of user properties. The 'Level' property is highlighted.

NAME & ADDITIONAL INFO	CONDITIONS
My Segment	[User Prop: Select item...] x
Frequency/F0 (None) How Often player makes purchases	[User Prop: ...] Purchase <= 0
Frequency/F1 (Daily) How Often player makes purchases	[User Prop: ...] Purchase ∈ [0 , 2)
Frequency/F3 (2-4 a Week) How Often player makes purchases	[User Prop: ...] Purchase ∈ [2 , 6)

Profiles

Profiles introduce a dedicated section for creating and managing user profiles within Balancy.



Creating a structured, singular profile for each user is advised to house all necessary data efficiently. This organization is crucial for maintaining well-structured data, crucial for game development and operations.

Creating Profiles

- **Unlimited Profiles:** While Balancy allows the creation of unlimited profiles, a single, well-structured profile per user is recommended for simplicity and efficiency.
- **Data Types Supported:** Supports all primitive data types (string, int, float, etc.), lists, and Data. "Data" refers to your custom Template with its own parameters, facilitating organized data management beyond simple key-value pairs.

Technical Limitations

- **First Layer Restrictions:** Primitive data types cannot be added directly to the first layer of your Profile; only Data parameters are permitted here.
- **Data Size Limit:** Each Data parameter, including all its nested parameters, must not exceed 150Kb. This constraint is particularly important to consider for parameters storing lengthy strings.

Best Practices for Profiles

Profiles are pivotal for storing comprehensive player progress and data. To optimize profile utility and efficiency:

- **Data Segmentation:** Segregate stored parameters by their relevance and function. For instance, your profile might categorize data into `QuestsProgress`, `GeneralInfo`, `Characters`, `Scenario`, `Level`, etc.

Ads

1. After placing a rewarded video or any other Ad, you can track the revenue you made from it using the following method:

```
Balancy.LiveOps.Ads.TrackRevenue(<ad_type>, <revenue>, <placement>);
```

Balancy supports 3 types of Ads:

- Rewarded
- Interstitial
- Custom

You can use the **Custom** ad type for any ads, which don't fit into Rewarded or Interstitial.

2. You can use the revenue earned or the amount of placed Ads in the [Conditional Logic](#):
 - a. Select the Primitive User Property condition.
 - b. Choose ads metric.

The screenshot displays the Balancy interface for configuring Conditional Logic. It shows two ad campaigns: 'WeekendEvent' and 'Test'. The 'Test' campaign is selected, and a search dropdown is open over the 'CONDITION' field. The search dropdown shows a search for 'ads' and a list of metrics including Revenue, Count, and Revenue Today under various ad types (None, SYSTEM > ADS INFO, SYSTEM > ADS INFO > AD REWARDED, SYSTEM > ADS INFO > AD INTERSTITIAL).

3. You can use Ads in [Visual Scripting](#) to run your campaigns. For example: After watching 3 **Rewarded Videos** the player gets a **Starter Pack Hero** offer:

Overrides

This section allows you to change parameters of the documents based on the conditions.

The screenshot displays the 'Overrides' management interface in the Balancy system. The main area is titled 'Overrides' and includes a '+ Create Override' button and a 'Save' button. A search bar is present in the top right. The interface shows a 'New Override' form with a 'NEW' status and an 'UPDATED: 7 MAY, 2023 12:07 PM' timestamp. The form contains a table with the following structure:

CONDITION	DOCUMENT	PARAMETER	NEW VALUE	DEF. VALUE	COMMENTS
[In Segment MaxPay/\$10]	Tiny of Gems	Price	Open		
	Chest	Icon	Storel...		Gems Pack 0 (\$1.99)

Below the table, there are buttons for '+ ADD ROW' and '+ ADD SECTION'. The left sidebar contains navigation options under 'CONTROL' (Deploy, Platforms, Data Structure, Localization) and 'LIVEOPS' (Game Events, Offers & Items, Scripts, A/B Tests, Overrides, Segmentation).

Examples:

1. You can change any **StoreItem** price during the weekend.
2. You can change Icons for items during the Halloween event.
3. You can change the script executing during the next event for paying customers.

Conditions

Conditional logic can be applied to any document. Currently, we use it to run [GameEvents](#). Any condition starts with one of 2 logical operators **And** / **Or**. Then you can add any other types of conditions. The conditions can be as complex and nested as you want.

- **Dates Range, Day Of The Week, and Time Of The Day** are conditions based on a specific time.
- **ABTest Condition** checks if the user was assigned to a specific A/B Test and Variant.
- **Active Event** triggers only if the specified **GameEvent** is active right now.
- **Segment Condition** checks if the user belongs to a specific Segment.
- **Revenue** checks if the user has generated a specific amount of in-app/ads revenue(count) during certain days.
- **Was Item Purchased** checks if a StoreItem was purchased at least once.
- **Was Product Purchased** checks if a Product was purchased at least once.
- **Primitive** allows you to use any custom [User Properties](#).

The screenshot shows two configuration panels. The top panel is for a 'WeekendEvent' (UPDATED: 7 MAY, 2023 10:55 AM). It has a TAG of 'Special Offers', a SCRIPT dropdown set to 'Select item...', and a TYPE of 'Tournament'. The CONDITION field is active, showing a search bar and a dropdown menu with options: LIVEOPS (ABTest, In Segment, Is GameEvent active, Revenue, Was Item Purchased), LOGIC (And, Not, Or), and SYSTEM. The bottom panel is for a 'Test' (UPDATED: 6 MAY, 2023 5:21 PM). It has a TAG of 'Special Offers', a SCRIPT dropdown set to 'Simple Script' with a 'View' button, and a CONDITION field set to 'None'.

- **Country** checks if the user is from the specified country.

Note: Please use this **Country** condition under the **System** category instead of getting the custom Country User Property from **Primitive** condition.

Push Notifications

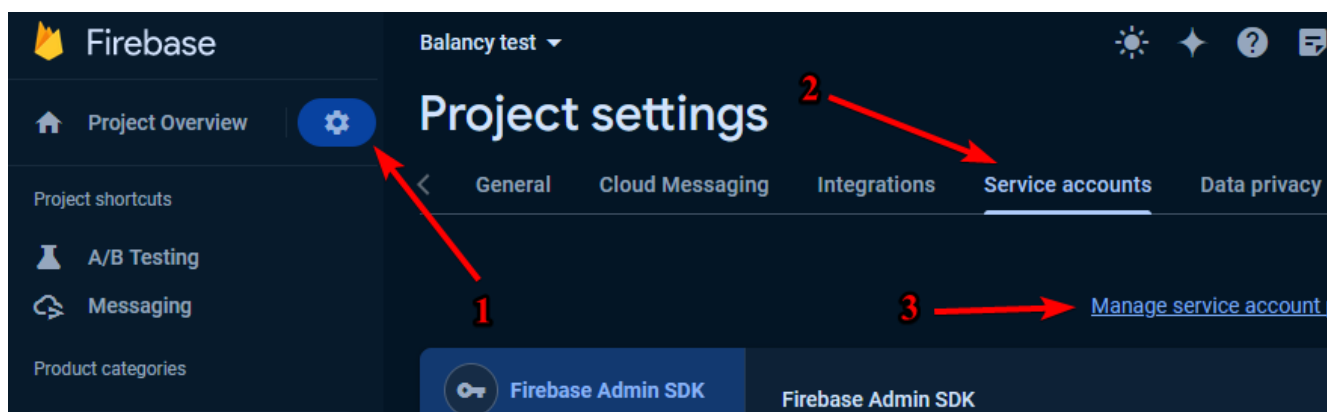
Push Notifications are in closed beta test!

Contact Balancy team to request access to the feature for your project.

Setup

First of all you need to add Firebase credentials into Balancy.

1. In Firebase console open integration page and manage service account permissions.



2. On the Firebase Cloud Messaging API page, click Enable.

Firebase Cloud Messaging API

[Google](#)

FCM send API that provides a cross-platform messaging solution to reliably deliver messages at no...

[MANAGE](#)
[TRY THIS API](#)
✓ API Enabled

3. Create additional role (or use existing one) with permission [cloudmessaging.messages.create](#). You can check if you already have this role on the analyzer page.

Section for Programmers

To receive important LiveOps events, such as new Offer activate/deactivate, A/B test started, Segment changed, etc... follow the next steps:

1. Create a new class inherited from interface `ISmartObjectsEvents`. You can use our example `SmartObjectsEventsExample`.

```
using Balancy.Data;
using Balancy.Data.SmartObjects;
using Balancy.Models.SmartObjects;
using Balancy.SmartObjects;
using UnityEngine;

namespace Balancy
{
    //TODO make your own version of this file, because the original file will be
    //overwritten after Balancy update
    public class SmartObjectsEventsExample : ISmartObjectsEvents
    {
        public void OnSystemProfileConflictAppeared()
        {
            Debug.Log("=> OnSystemProfileConflictAppeared");
            // System Profile is created and handled automatically. It contains Scripts
            // progress, active Events and Offers, information about A/B testing, Payments, Segmentations,
            // etc...
            // Balancy.LiveOps.Profile.SolveConflict(ConflictsManager.VersionType.Local);
            Balancy.LiveOps.Profile.SolveConflict(ConflictsManager.VersionType.Cloud);
        }

        public void OnNewOfferActivated(OfferInfo offerInfo)
        {
            Debug.Log("=> OnNewOfferActivated: " + offerInfo?.GameOffer?.Name +
" ; Price = " + offerInfo?.PriceUSD + " ; Discount = " + offerInfo?.Discount);
        }

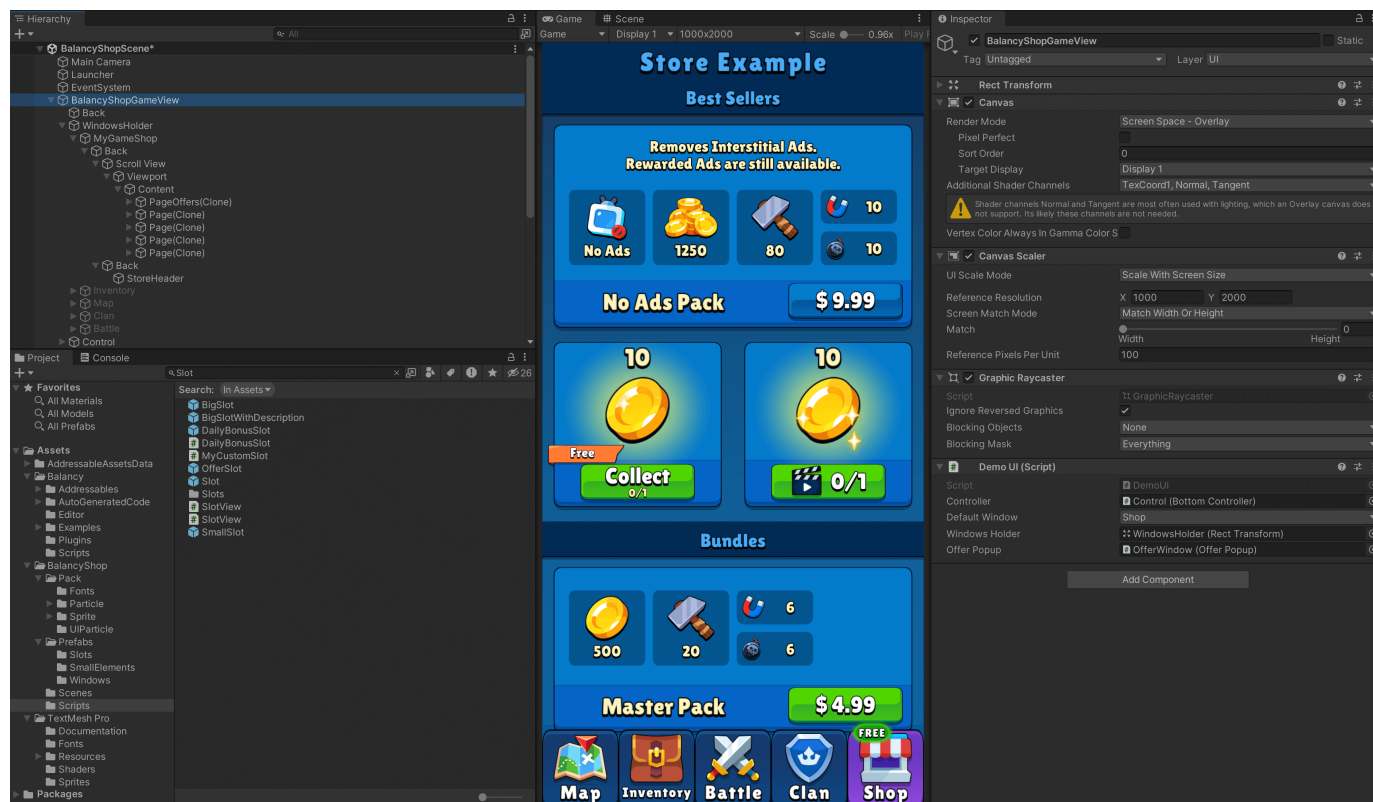
        public void OnNewOfferGroupActivated(OfferGroupInfo offerInfo)
        {
            Debug.Log("=> OnNewOfferGroupActivated: " + offerInfo?.GameOfferGroup?.Name);
        }

        public void OnOfferDeactivated(OfferInfo offerInfo, bool wasPurchased)
        {
            Debug.Log("=> OnOfferDeactivated: " + offerInfo?.GameOffer?.Name + " ;
wasPurchased = " + wasPurchased);
        }

        public void OnOfferGroupDeactivated(OfferGroupInfo offerInfo, bool wasPurchased)
        {
            Debug.Log("=> OnOfferGroupDeactivated: " + offerInfo?.GameOfferGroup?.Name +
" ; wasPurchased = " + wasPurchased);
        }
    }
}
```


Template: Game Shop

Integrate a universal game shop into any game using this versatile template.



Getting Started

Follow these steps to implement the shop in your project:

1. **Add Shop Template:** Integrate the **Shop Template** into your Balancy project.

Game Shop: Prefabs and Classes Overview

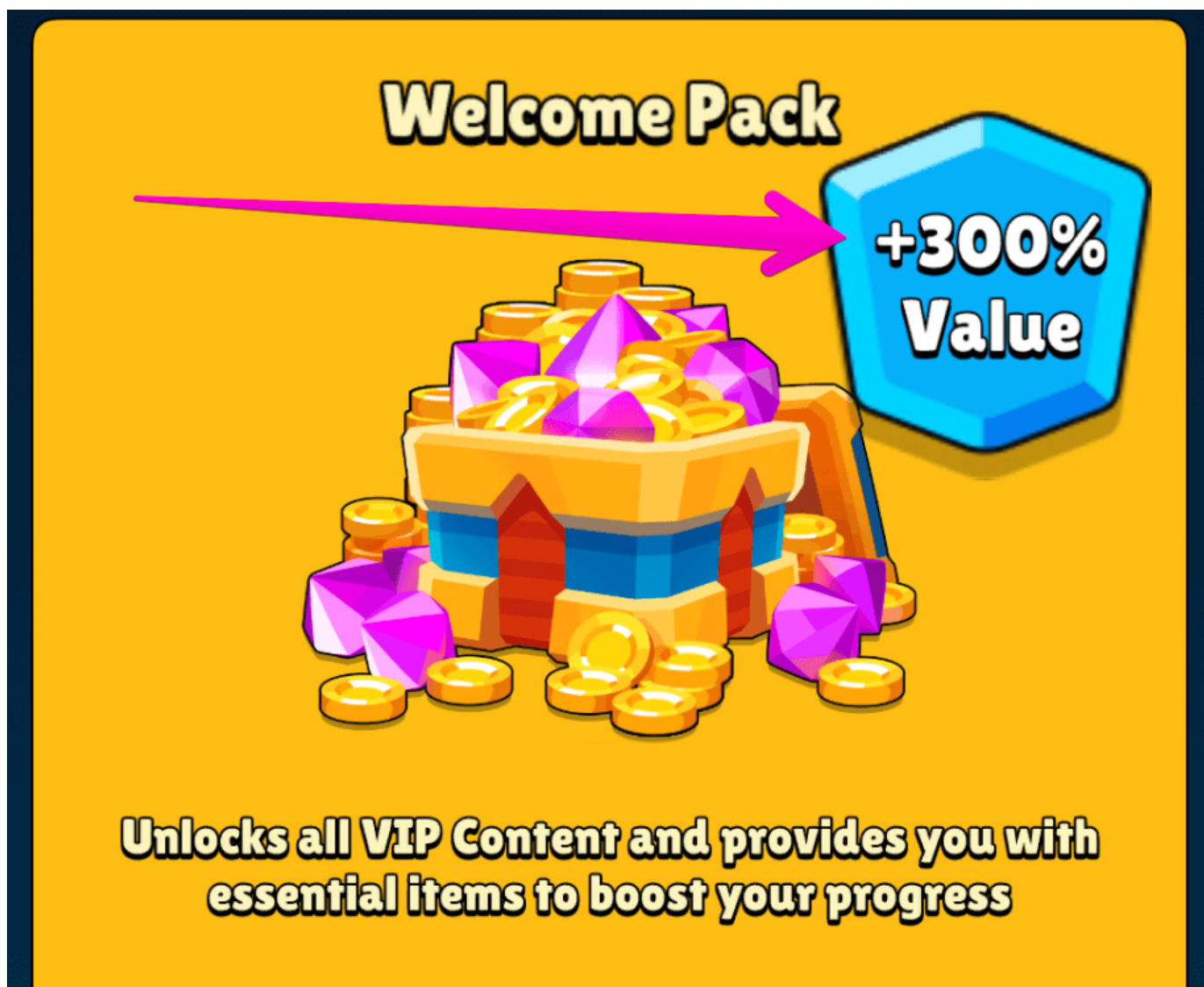
Explore the pre-built components and classes in Balancy's game shop, designed for optimal integration and customization.

Balancy Templates

Badge

- **BalancyShop.Badge**: Contains Sprite and Text information for badges.

The Badge is used as a parameter in [UIStoreItem](#) The badge can be displayed in the [SlotView](#) or [OfferPopup](#):



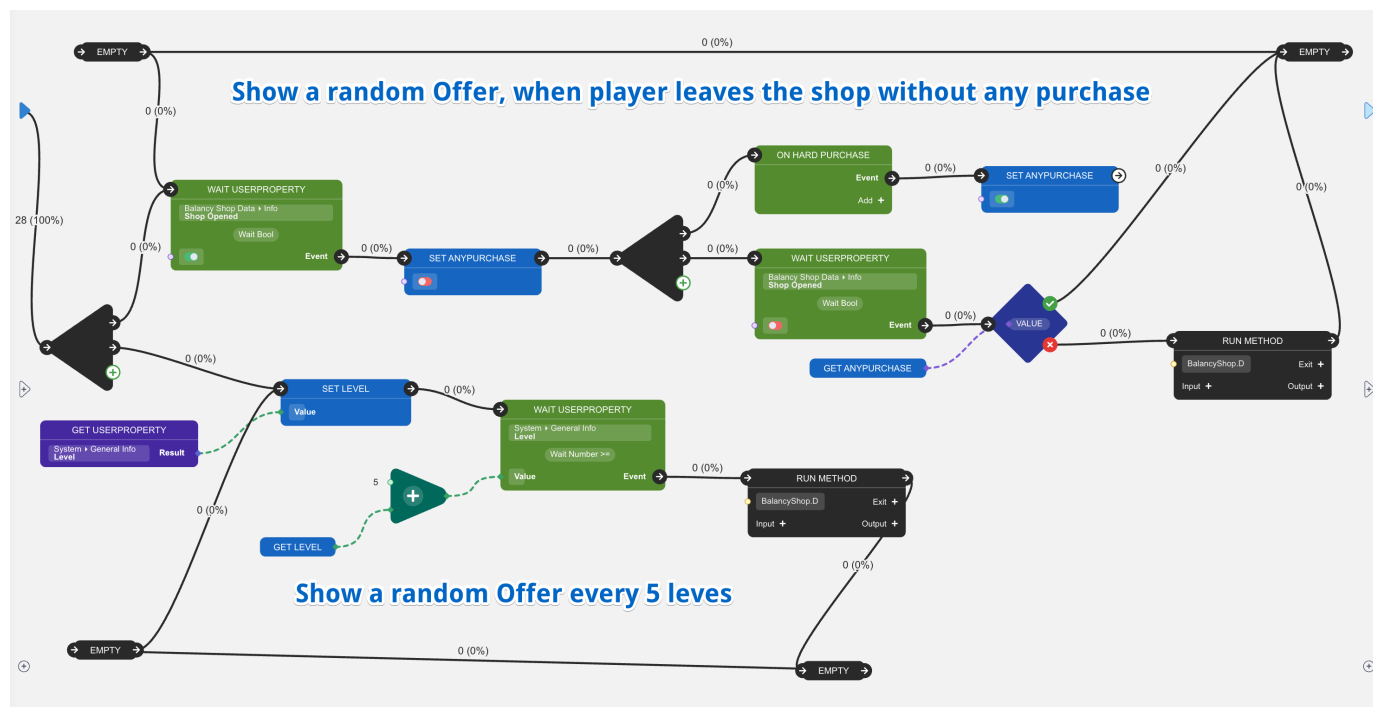
Scripts for Shop

Show Offer Popup

A script that displays a random Available Offer as a Popup under these conditions:

1. **Level-based Trigger:** The popup appears every 5 levels.
2. **Shop Exit Trigger:** Activated when a player exits the shop without making any purchases.

This setup is just an example, and you are encouraged to modify the logic to suit your game's needs. Consider implementing a cooldown between popups to avoid overwhelming players with frequent offers.



The **RUN METHOD** node in the script triggers a C# method using reflection. Specify the full path to the static method you wish to execute in the input port, such as: `BalancyShop.DemoUI.ShowRandomOffer`.

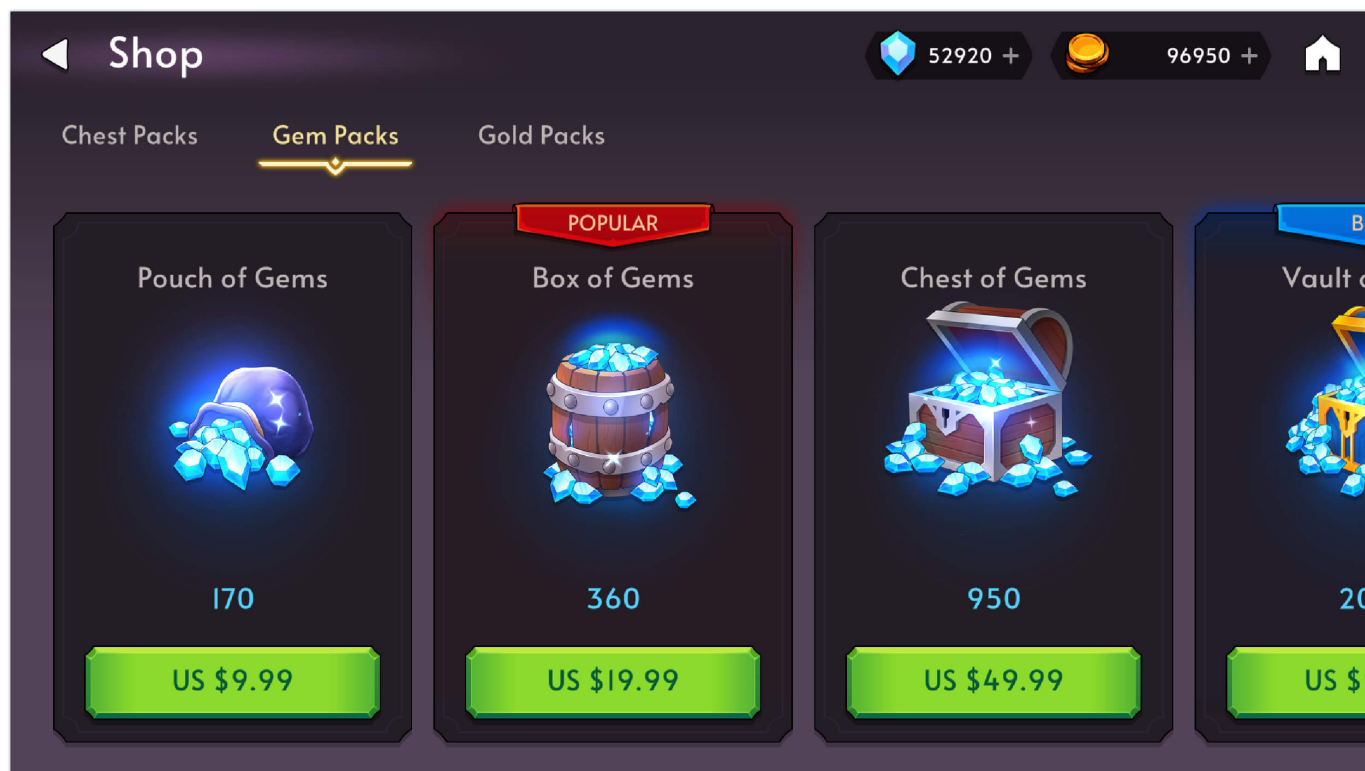
C# Method Implementation:

Here's the complete code for the `ShowRandomOffer` method:

```
using Balancy.Data.SmartObjects;
using Balancy.Models.GameShop;
using UnityEngine;

namespace BalancyShop
{
    public class DemoUI : MonoBehaviour
```


Balancy Example



Description

This Demo shows how to create a Store with Segmentation and launch personalized Special Offers.

How to start

1. In the Balancy Dashboard, click on the **Templates**.
Screenshot
2. Clone Project in Balancy.
3. [Deploy](#) the created project.
4. Clone the [Git Project](#).
5. Open the project in Unity.
6. Launch the scene **Assets/ExampleGame/Scenes/SampleScene**.

Template: Clash of Clans

[Clash of Clans](#) is a popular mobile strategy game developed and published by Supercell. It was first released for iOS devices in August 2012, followed by an Android release in October of the same year. The game has since become a worldwide phenomenon, with millions of active players across the globe.

In the game, players build and upgrade their village, train troops, and battle against other players to earn resources and increase their rank. The game's success is partly due to its addictive gameplay and regular updates, which keep players engaged and constantly improving their strategies.



Clash of Clans has received numerous awards and has been downloaded over 500 million times from the Google Play Store alone. It has also spawned a massive esports scene, with professional players and teams competing in tournaments for cash prizes.

Store Template

Are you looking to create a store system like Clash of Clans in your game? Look no further than Balancy. Our team has thoroughly analyzed and reconstructed the data used by Clash of Clans to create their store system, and we've incorporated that knowledge into our platform. With Balancy, you can see how resources work and what formulas are used to determine their prices.

Template: Merge Game



Merge games - a captivating sub-genre of puzzle games where players combine identical items to create new, more advanced ones. These games have grown exponentially, captivating audiences with their intuitive mechanics and rewarding progression systems.

At the core of merge games are intricate items with unique attributes and functionalities. Some items can boost your progress, while others might be split, downgraded, or transformed in diverse ways. Given this complexity, developers often need help storing and managing these myriad items with varying settings.

To assist with this, we've crafted a comprehensive template to address many common questions and challenges you might face.

For more insights, don't hesitate to explore an article on our [blog](#). While it discusses a solution tailored for survival games, the underlying principles are versatile and can be adapted for merge games.

Dive in, and equip yourself with the knowledge to take your merge game development to the next level!

Items

Items are the core of any merge game. Each item has a lot of parameters, but the most important one is **Item Effects**, here you define all unique features of each item.

Template: Wheel Of Fortune



The Wheel of Fortune mechanic in games involves a spinning wheel divided into different sections, each representing a prize, reward, or action. Players spin the wheel, and the section the wheel lands on determines what the player receives or has to do next. This mechanic is often used in games as a way to distribute random rewards and can add an element of luck and excitement to the gameplay.

Wheels

The Wheel Of Fortune template allows you to create multiple configs. You can provide different rules for each of the configs. The configs are called Documents in Balancy. One of the parameters in Wheel of Fortune is **Condition**. We are going to use it to give different configs for players, depending on their level, but you can come up with any other segmentation.

ID	Name	Condition	Drop	Spins Per Day	Updated
506	Start Wheel	[User Prop: System > General Info > Level < 10]	[505] Test Item x 5 Weight 100 [505] Test Item x 10 Weight 50 [505] Test Item x 20 Weight 20 [505] Test Item x 50 Weight 10 [505] Test Item x 100 Weight 2	3	26 Aug, 2023 5:45 PM
539	End Game Wheel	[User Prop: System > General Info > Level >= 10]	[505] Test Item x 20 Weight 20 [505] Test Item x 50 Weight 10 [505] Test Item x 100 Weight 2	1	26 Aug, 2023 5:45 PM

Battle Pass System

The Battle Pass system is a popular monetization strategy in video games, especially in free-to-play online multiplayer games. It consists of a series of tiers with rewards that players can earn by completing specific challenges or tasks within the game.

Typically, there are two tracks in a Battle Pass: a free track and a premium track. The free track is available to all players and offers basic rewards, while the premium track offers more valuable and exclusive rewards but requires the player to purchase the Battle Pass with real money or in-game currency.

Players progress through the tiers of the Battle Pass by earning experience points (XP) or completing specific challenges. The more you play, the more rewards you unlock. Once the season ends, a new Battle Pass is released with a new set of rewards and challenges.

This system encourages continuous play and engagement with the game, as players are motivated to complete challenges and earn rewards before the season ends. It also provides a way for developers to monetize their game while offering value to the players.



In the example image above, you can see the two tracks of the Battle Pass. The top track is the premium track, which includes exclusive rewards such as skins, emotes, and in-game currency. The bottom track is the free track, which includes basic rewards such as experience points and consumable items. Players can progress through the tiers by earning experience points or completing challenges.

Interstitial Ad Strategy

Incorporating an effective ad strategy is pivotal for both enhancing user experience and ensuring monetization for developers. Drawing inspiration from a study by [SuperSonic](#), this template introduces a dynamic interstitial ad strategy.

The core principle is to show ads infrequently at the initial stages, specifically every 90 seconds for users up to level 10. As players become more engaged and progress through the levels, the time between ads reduces. By the time players reach level 30, they encounter an ad every 30 seconds. Implementing this strategy has been shown to result in an approximate D7 ARPU increase of ~25%.

Features of the Template

Game Event: Trigger for the Infinite Script

A straightforward Game Event is employed, which instantaneously activates the Script.

Infinite Script Logic

This Script remains vigilant for changes in a player's level. Once a change is detected, it recalculates the **InterstitialAdsPeriod** value, adjusting the frequency of ads displayed.

Ad Watchers Conversion




This Template's purpose is to convert your loyal customer, who watch a lot of rewarded ads, to your paying customer.

We are tracking how many rewarded ads a user has watched. Once they have watched 20 rewarded ads, they were exposed to a special offer at a good price. If they convert, we simply finish the script, our goal is achieved. If they don't make a purchase, we wait until they have watched 50 ads and make one more special offer for the same price, but higher value. We kept increasing the value until they finally convert into their first purchase in the game. And as you know, once users make their first purchase, they are more likely to make the second, third and so on.

What Template Includes

Game Offers

3 Offers with different value, but the same price of \$0.99. Each next offer is more valuable than the previous one.

Golden Offer 100				DURATION: 3H	LIMIT: 1	UPD: 14 MAY 2023
SPRITE	DESCRIPTION	STORE ITEM	REWARD			
	This Offer is granted to players who watched 100 rewarded Ads.	<ul style="list-style-type: none"> Golden Offer 100 Gold Offer (\$0.99) 	[486] Gold x 300			
Golden Offer 50						
Golden Offer 50				DURATION: 3H	LIMIT: 1	UPD: 14 MAY 2023
SPRITE	DESCRIPTION	STORE ITEM	REWARD			
	This Offer is granted to players who watched 50 rewarded Ads.	<ul style="list-style-type: none"> Golden Offer 50 Gold Offer (\$0.99) 	[486] Gold x 200			
Golden Offer 20						
Golden Offer 20				DURATION: 3H	LIMIT: 1	UPD: 14 MAY 2023
SPRITE	DESCRIPTION	STORE ITEM	REWARD			
	This Offer is granted to players who watched 20 rewarded Ads.	<ul style="list-style-type: none"> Golden Offer 20 Gold Offer (\$0.99) 	[486] Gold x 100			

Conversion Script

The script tracks the amount of rewarded ads watched and shows Offers at 20/50/100 ads.


A/B Test + Starter Pack


The template has a Script that launches a Starter Pack offer via A/B Test.

What Template includes

Game Offers

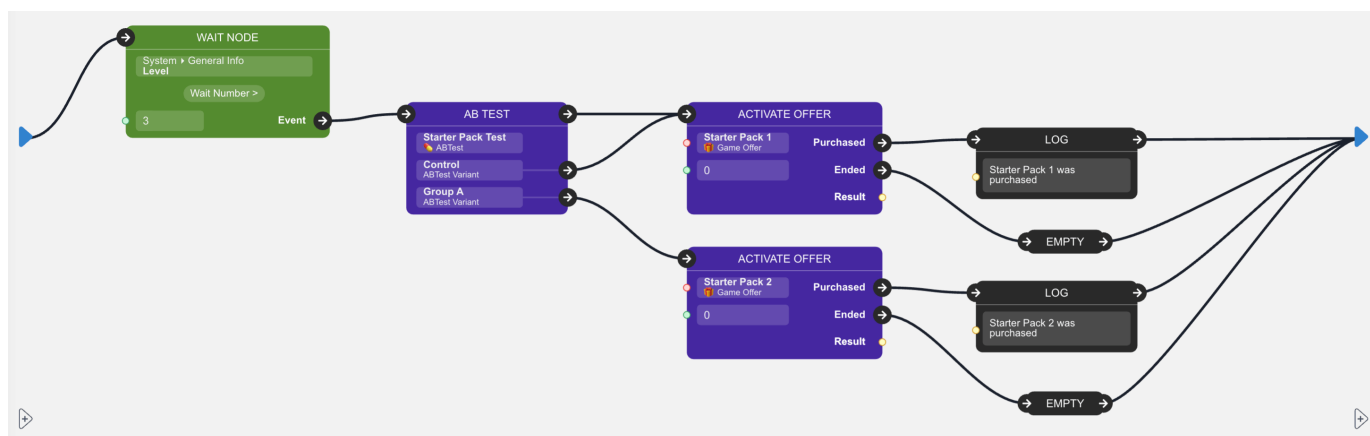
2 Offers with different value and price.

Starter Pack 1 UPDATED		DURATION: 3H LIMIT: 1 UPD: 14 MAY 2023	
SPRITE	DESCRIPTION	STORE ITEM	REWARD
		<ul style="list-style-type: none"> Starter Pack 1 Starter Pack 1 (\$4.99) 	[477] Gold x 100

Starter Pack 2 UPDATED		DURATION: 3H LIMIT: 1 UPD: 14 MAY 2023	
SPRITE	DESCRIPTION	STORE ITEM	REWARD
		<ul style="list-style-type: none"> Starter Pack 2 Starter Pack 2 (\$9.99) 	[477] Gold x 200

A Script

The script waits until player reaches level 3 and then Activates one of the offers.



Game Event

A simple event only launches the Script.

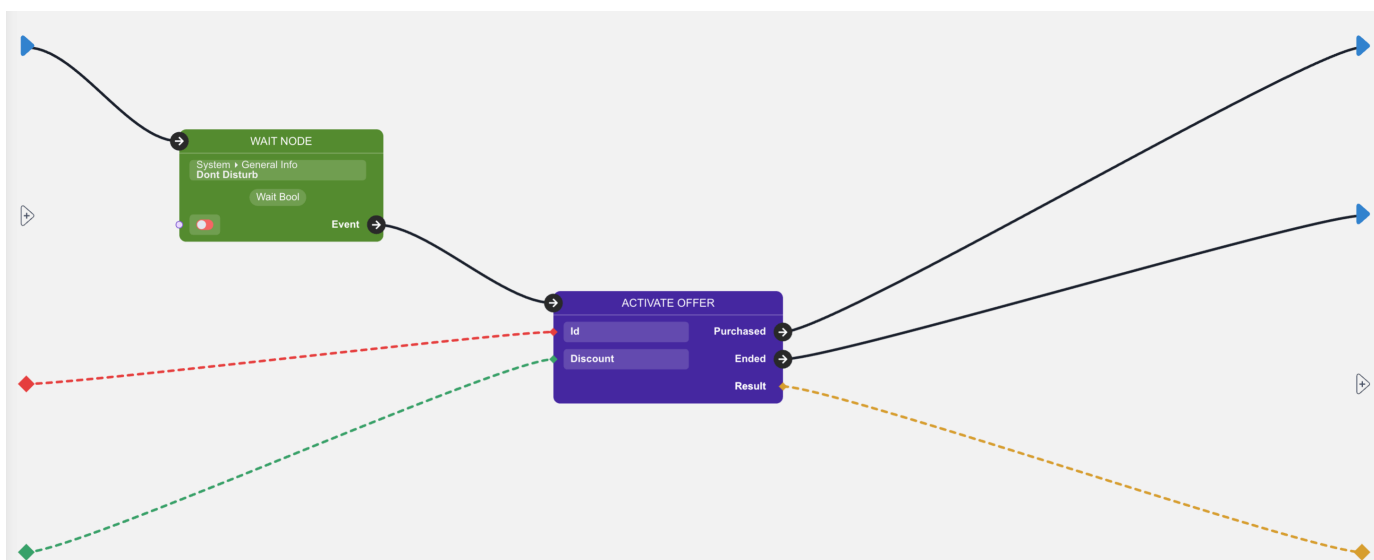
Don't Disturb

An example of usage of the Don't Disturb flag. With this flag, you can prevent Offers from appearing during the battle, or Activate offers only on the main screen.

What Template includes

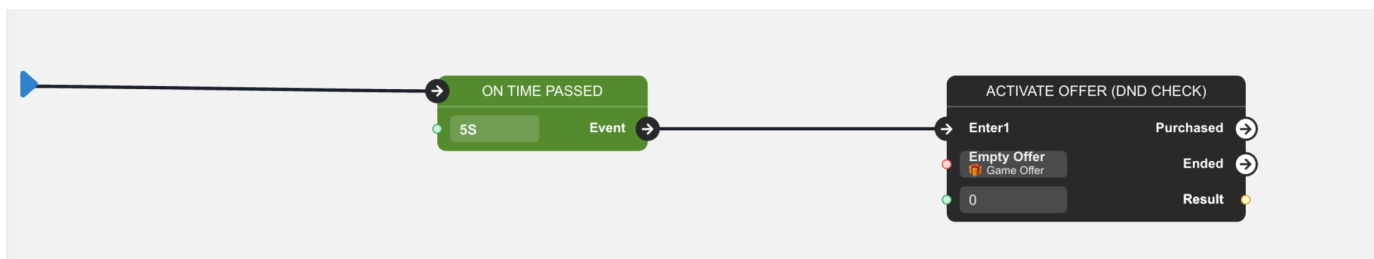
Activate Offer Script

A new version of a Script, which activates an offer, only when the **DontDisturb** flag is false.



Conversion Script

This script activates the offer using the script above, after 5 seconds of the gameplay.



Game Event

A simple event only launches the Script.

Content Management System

CMS (Content Management System) is an essential part of Balancy. It is used for creating data structure and editing the data. Balancy automatically delivers the newest data to the app and parses it to the convenient auto-generated code, so you can easily access it.

How Does it Work?

1. You can add all types of objects your game has: weapon, item, construction, monster, hero, location, etc...
2. You can add as many documents as possible for each type of object. Each document represents a unique weapon, item, construction, etc...
3. Open your project in Unity and start code generation requests. Balancy will automatically generate the code based on the data you provided.
4. Once the game is launched, all the game data is delivered to the game and already mapped to the generated code.
5. Your programmer has direct access to your game's items, weapons, and other objects. He doesn't have to write any code for downloading or parsing.
6. All changes in Balancy will be automatically synchronized with the game on launch. You must [deploy changes](#).

I

Templates

Template describes the structure and behavior of your game object (item, monster, construction,...). As a programmer, you can think of it as a class. The template has to have a unique name and may contain a set of parameters.

1. Open the **Data Structure** section, you'll land on the **Templates** subsection by default. Click on the **Create Template** button.

Data Editor Tricks

























Tables

Columns pining

You can pin columns into left or right.

Templates / ❤️ Card / Documents

❤️ Card + Create New

<input type="checkbox"/>	 Name  	ID		Asset	
<input type="checkbox"/>	 Sort ascending	1729		 .../Wolpertinger.png	
<input type="checkbox"/>	 Sort descending				
<input type="checkbox"/>	 Stick left	1732		 .../FlameImp.png	
<input type="checkbox"/>	 Stick right				
<input type="checkbox"/>	 Hide				
<input type="checkbox"/>	Warlock 	1733		 .../Webspinner.png	
<input type="checkbox"/>	Battlefiend 	1738		 .../Battlefiend.png	

Improved editing

You can navigate through cells via arrows, tab and enter buttons. Also, it's possible to copy-paste cells' values.



Packages

Balancy has packages/templates system similar to same systems on other platforms (e.g. Unity packages system). It allows you to install existing public packages and [templates](#) made by our team.

The most known and used public package is LiveOps package. It allows you to use [additional features](#) like segmentation, A/B Tests, game events, etc.

Private Packages

You can create your own private packages to use them in your projects. Open **Packages** page from Project Settings menu and press "+ Create" button.

	Name	Installed version	Version
	Core	6	6
	LiveOps	128	128

After that create a new one.

One project one package

Currently one project can have only one private package for export. So if you are planning to create several private packages, do it in separate projects.

In the right corner chose "Private" and "Default".

Code Generation

Why do I need it?

It's a good question because many developers like to rewrite such things in every project.

Here are some reasons for you to consider:

1. Why would you spend your precious time on a monotonous job? Such things should've been automated a long time ago.
2. Human mistakes excluded. A game designer or programmer often misspells a word, and parsing doesn't work as expected. Such a problem might take some time to be found.
3. Whenever a game designer changes a parameter or a Template, all of that will be instantly reflected in the code after the generation. A programmer will remember to apply those changes.
4. Code generation is tightly connected with other Balancy excellent features, like [Localization](#). Using them all together gives your team a huge boost.
5. You can remember JSONs and how to parse them. Balancy will do that for you, so you can only work with convenient classes.
6. When a document refers to another document, developers usually use some ID to create those links. Balancy will do that for you. It automatically resolves all links and gives you direct access to the Documents you expect.

How to generate code

1. In Unity, select Tools ► Balancy ► Config, enter your email and password for balancy.dev, and press **Generate Code** button.
2. It might take some time, depending on your connection and the number of Templates you have.
3. Generated classes will be placed in Assets/Balancy/AutoGeneratedCode.
4. Please **DO NOT** change anything in this folder because your changes will be overwritten with the next generation.

How does it work inside?

Balancy server-generated JSON files based on your Documents and puts them in the CDN storage. Balancy plugin automatically checks for the updates of those JSON files and downloads only updated files. After that, it parses the data from JSON to the generated classes and finds all dependencies. The programmer doesn't have to download, parse, or understand JSON. There is also no need to find any links if any of the Documents refers to

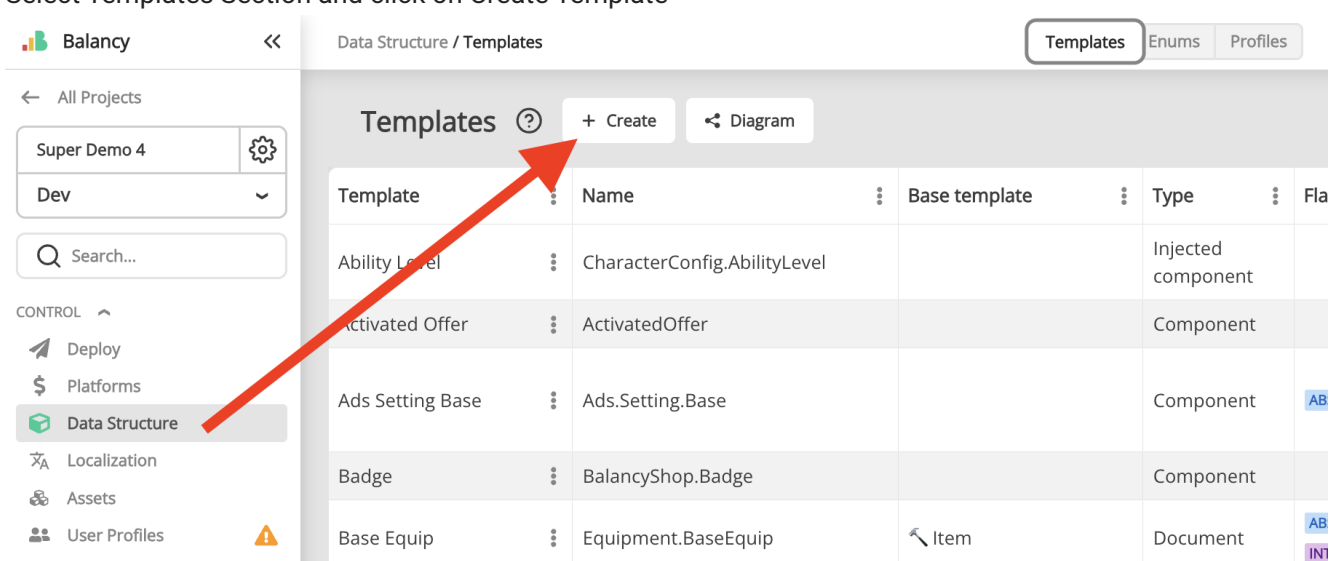
CMS Usage Example

In this example, I will show you how you can create a simple craft logic for your game. We will need just several Templates: Item, Recipe, and Ingredient. I assume you've read and implemented [the basics](#) for your game.

Here is [the video](#) of this tutorial.

Templates preparation

1. Select Templates Section and click on Create Template



The screenshot shows the Balancy CMS interface. On the left, there is a sidebar with a 'Data Structure' menu item highlighted. The main area displays the 'Templates' section with a table of existing templates. A red arrow points to the '+ Create' button in the top right of the table area.

Template	Name	Base template	Type	Fla
Ability Level	CharacterConfig.AbilityLevel		Injected component	
Activated Offer	ActivatedOffer		Component	
Ads Setting Base	Ads.Setting.Base		Component	AB
Badge	BalancyShop.Badge		Component	
Base Equip	Equipment.BaseEquip	Item	Document	AB INT

2. Set **Name** as "Item" and leave other fields as default.

3. Add the following parameters:

- **Name:** (Type: String, Use in display name - YES)
- **Description:** (Type: String)

4. Create another Template for Ingredients. We'll make it a Component for convenience.

- **Name:** Ingredient
- **Type:** Component

5. Add the following parameters:

- **Item:** (Type: Document, Reference Template: Item)
- **Count:** (Type: Integer, Default value: 1)

6. The last template we going to need is a Recipe. Set **Name** as "Recipe" and leave other fields as default.

7. Add the following parameters:

- **Item:** (Type: Document, Reference Template: **Item**)

Deploy

Whenever you change a Template or a Document in Balancy, it's saved on our servers to be available for your end users. You can compare all the changes you make in Balancy with commits in GIT, the only difference being that other Balancy users can see your commits.

When you are ready to publish (like Push in git) your changes for the end users, open the **Deploy** section in Balancy and click **Deploy** button. The process takes a few seconds and once it's completed you can launch your game in Unity and get all the updated data.

Bear in mind that you are Deploying only to the active Branch.

Redirections (Obsolete)

Obsolete section

Redirections are supported only for the old versions of plugins in the games released before **Branches** system update. New plugin doesn't require using redirections.

Sometimes you need to make a production build but use staging balance. For example, during store review or for partial rollout. Until we have branches functionality (like in Git), you can use redirections to point the game build to download balance from another environment.

Environment

We provide each game with 3 Environments: Development, Stage, and Production. It's a standard and commonly used approach.

1. **Development** used during the development process. All new features and bug fixes are created here.
2. **Stage** can be skipped by Indie developers. Big companies usually use it with their own QA department. This environment is used for testing all the features which were created before. A separate environment for testing is helpful because it doesn't require the development team to stop.
3. **Production** is where all your live clients play.

Workflow

1. New features are being developed and balanced in the **Development** environment.
2. Open the Environment section and migrate your **Development** data to the **Stage** environment once ready.
3. It'll erase all the changes you made in the **Stage** environment and copy everything from the **Development**.
4. Give the build connected to the **Stage** environment to your QA.
5. Once QA approves the build, transfer the data from **Stage** to **Production** and publish your game build (connected to the **Production**) in the stores.

Changing The Environment

You should try to avoid changing the environment in Balancy. In the best-case scenario, you must work in the **Development** and transfer the data to other environments.

However, there are situations when you need to change something in the **Production** environment.

Let's say you have published your game, and your users are playing in the **Production** environment. Then you find some bug in the balance, which you want to fix asap. Your team has already prepared many changes in the **Development**, so you can only migrate everything to the **Production** by updating the build. So the solution here is to switch to the **Production** environment in Balancy, fix your balance, and Deploy the changes. That's it! Remember to make the same changes in the **Development**.

How To Connect To The Proper Environment

We usually use [Define Symbols](#) to deal with different environments:

```
Balancy.Main.Init(new AppConfig
{
    ApiGameId = <API_GAME_ID>,
```


Branches

Introducing Branches & Versioning system, the workflow which allows you to work with your game data/configs in Balancy in a Git-like way. Old environment-based workflow had a lot of limitations.

With branches, you can:

- Work on different game features simultaneously: one game designer can work on battle pass in feature/battlepass branch, while another one - on in-game quests in feature/quests branch without interfering with each other. And later they can merge all changes into main branch.
- Move changes between branches without replacing or discarding changes in target branch.
- Version your releases to run multiple builds with different game balance simultaneously.

Migration to branches

Read [additional notes](#) to migrate from the old environments-based workflow.

Branch Creation

Currently selected branch is shown below Project Name.

Current Branch

All other existing branches are available via dropdown menu from this selector. On the Deploy page you can create a new branch from the current one.

New Branch

The following UI will allow to set up the new branch settings:

Branch creation

It will be the copy of the current branch. After that you can make changes in this branch without being afraid of breaking something in the original branch.

Typical Scenarios

Let's see some typical scenarios which would be possible to implement from now on.

Display Format

Imagine you have a very complex structure of conditions based on Components. It should look something like this:

ID	Name	Condition	Description
22	Special Offer	And: Conditions [Field >=: Field PlayerLevel Value 5 Field <: Field Gold Value 1000]	It supposed to convert our users to paying customers
23	Gold Offer	And: Conditions [Or: Conditions [Field Range: Field PlayerLevel Min 0 Max 3 Field >=: Field Gold Value 200] Or: Conditions [Field ==: Field PlayerLevel Value 10 Field Range: Field PurchasesCount Min 0 Max 2 Field >=: Field GoldCoins Value 99]]	It should increase ARPPU for the day 7

You can easily change how your components are displayed without changing the logic.

ID	Name	Condition	Description
22	Special Offer	&& [PlayerLevel >= 5 Gold < 1000]	It supposed to convert our users to paying customers
23	Gold Offer	&& [[0 <= PlayerLevel <= 3 Gold >= 200] [PlayerLevel == 10 0 <= PurchasesCount <= 2 GoldCoins >= 99]]	It should increase ARPPU for the day 7

1. Open Template, which displays the view you want to change.
2. Select **Use custom display format?**
3. **Display format** becomes available for editing
4. Use the following code to customize the display:

Name	Description
{template.displayName}	Display Name of the component will be placed instead

Localization

We let you import/export and work with all your localization in the Balancy.



When you create a new parameter, instead of selecting type **String**, make it **Localized String**. Whenever you try to access this parameter in the code, it'll instantly give you an already localized value.

Let's say you have a pair "Key": "LocalizedValue":

1. You create a new parameter with the type **Localized String**.
2. In the document, you create a new key or choose existing one.

Templates / Items / Documents

Items + Create New

ID	 Name	Amount	Price	Updated
239	Select item... 	1	5	19 Feb, 2023 4:12 PM
240	<input type="text" value="search..."/>	2	29	19 Feb, 2023 4:12 PM
241	Redgill Razorjaw: Redgill Razor...	1	9.99	19 Feb, 2023 4:12 PM
242	Bulwark of Azzinoth: Bulwark ...	1	19.99	19 Feb, 2023 4:12 PM

- Slam: Slam
- UI
- InApp1Name: Gems
- InApp1Desc: {0} Gems
- InApp2Name: Gem Sack
- InApp2Desc: {0} Gems
- InApp3Name: Big Gem Sacks

3. When you access your parameter from the code, you will get **LocalizedValue** instead of **Key**.

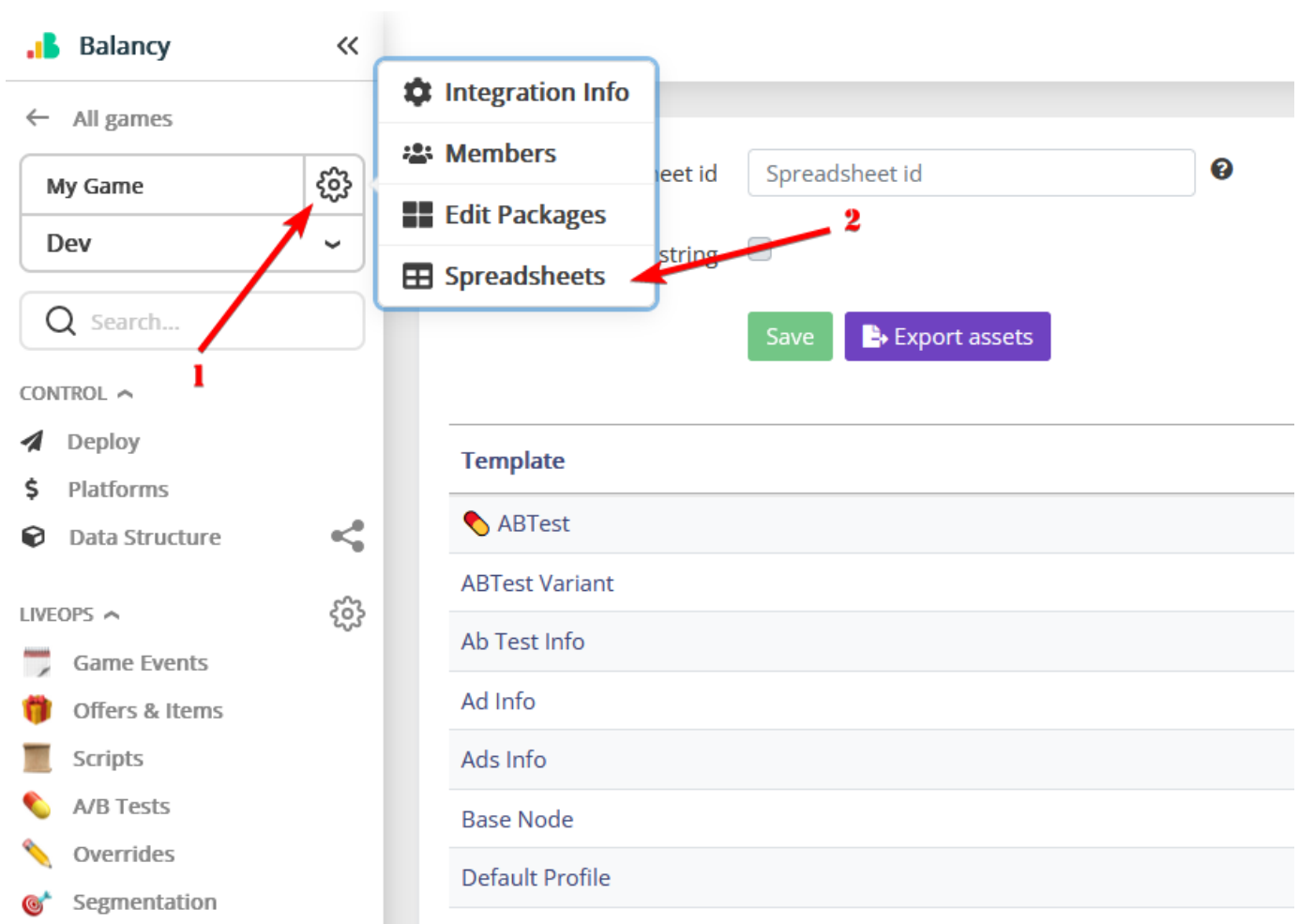
Spreadsheets

You may need to do additional data manipulation in Google Spreadsheets. You can set up import from them into Balancy and export from Balancy.

Suppose you plan to use Balancy for an existing game in which balance is stored in Spreadsheets. In that case, this feature helps import all the data into Balancy without filling everything manually.

Setup

1. Give write and read access to the service account by adding mail **google-sheets-api@balancy-334816.iam.gserviceaccount.com**.
2. Open the Spreadsheets settings window:



1. Copy the id from the address bar.

Updates

Balancy ensures that your game is always up-to-date by automatically synchronizing with the latest data. Understanding how updates work will help you manage your game more effectively and avoid potential issues.

How Balancy Updates Work

- **Content Delivery:** Balancy leverages CDN technologies (Cloudflare and Cloudfront) to distribute game configurations globally.
- **Optimized Updates:** Updates are structured around [Templates](#). Each template's documents are grouped into a single config for efficient updating. For instance, adding a new GameEvent only updates the GameEvents config.
- **Document Changes:** Any modifications to a [Document](#) mark its Template as "Dirty."
- **Deploying Changes:** Deploying updates all Dirty Templates and increments their version. Be aware of cross-environment version synchronization:
 - After synchronizing environments via migration, if the GameEvents template is at version 5 and changes are made in the Dev Environment, deploying will increase the version to 6.
 - Migrating this updated data to the Stage Environment also updates its version to 6.
 - If changes are made directly in the Production Environment, the Game Events version could jump to 7 upon deployment.

Client-Side Plugin Behavior

Scenario Without Pre-downloaded Data

1. **Initial Check:** At startup, Balancy compares locally cached configurations with the latest CDN versions, downloading updates as needed.
2. **Mapping:** Once synchronization completes, the updated configs are mapped to their respective classes.

Scenario With Pre-downloaded Data

This scenario requires careful handling to prevent issues. Follow these [best practices](#) for guidance.

1. **Version Checks:** Balancy compares the CDN configurations against those cached in Persistent Data and pre-downloaded in Resources. It skips downloading a config if its version is up-to-date.
2. **Final Mapping:** Configurations with the highest version from either Resources or Cache are selected and mapped to classes.

Authorization

Balancy's authentication methods are subject to updates to enhance versatility and, for certain platforms, processes may be automated for convenience.

Understanding Authentication

Balancy initializes with automatic user authentication via the device's unique ID, associating it with a guest account. This mechanism ensures persistent user progress, even across reinstallations, provided the device ID remains consistent.

Users have the option for further authentication methods, such as email or social media platforms, typically managed through FireBase. Post-authentication, these accounts can synchronize with Balancy using `Balancy.Auth.WithName`.

Authentication Scenarios:

1. **Switching Accounts:** To transition between Balancy accounts, follow these steps:
 - a. Sign out from the current Balancy session.
 - b. Utilize `Balancy.Auth.WithName` for re-authentication.
2. **Linking New Authentication Method:** Invoking `Balancy.Auth.WithName` can link a new authentication method. However, outcomes vary:
 - a. Should Balancy recognize it as a new authentication, the link is immediate.
 - b. If another user is already linked to this authentication, a **conflict** arises. You must then choose between the local profile or the one on Balancy's servers. Selection dictates whether the new method replaces the current link or if the local account updates to match the server's data.

An example of the flow if the same account is playing on two devices:

1. The new device creates a new account
2. When you do `Balancy.Auth.WithName` (non existing account), the auth method is connected to the existing account
3. If you launch the game on a new deviceID, the second account is created
4. If you try to authenticate with the same name as in step 2 (account already exists), the **conflict** appears `OnSystemProfileConflictAppeared`.
 - a. If you choose `Local`, the `Balancy.Auth.WithName` connects to the second deviceID and disconnects from the previous user.

Payments

Setup

1. Open the **Platforms & Products** page from Project Settings menu and add all the information about the Platforms where the game is available. It's required to validate the purchases.
2. Open the **Products** tab of Platforms & Products page and fill in all the information about your game's products. In most cases, you need the main table. However, if you have a different product ID, Name, or Price for other platforms, you can use the **override** section for each of the platforms.

Products are global

You have single global list of products, like in Apple Store or Google Play. Each environment uses the same list.

Make changes with care

Make changes with care, because it could affect your current players.

Products versioning

If you add/delete products on the products page, users will see these changes even without deploy.

Make changes with care

Be specifically care when you delete products.

Where to find needed data for validation for specific platforms?

Google Play

For Android, we need a License key.

1. Open Play Console and select the app you want to find the license key for.
2. Go to the [Monetization setup](#) page (Monetize ► Monetization setup).
3. Your license key is under «Licensing».
4. To properly check if the test account made the payment, you need to give us access to [service account](#).
 - Create a service account.
 - Link it to the google play dev account.

Requests, Response Data, and Errors

All Balancy methods are asynchronous with a callback as an argument, invoked once the method is complete. The callback parameter is inherited from `ResponseData`.

```
public class ResponseData {
    public bool Success;
    public Error Error;
    public object Data;
}

public class Error {
    public int Code;
    public string Message;
}
```

Response Example:

```
{
  "success": true,
  "error": {
    "code": 1,
    "message": "unknown error"
  },
  "data": //some data depending on the request
}
```

You need to check if `Success` is **true** to be sure that the request was successful and the `Data` is valid. Otherwise, you must read `Error` to understand what happened. Here is the list of errors that might occur:

```
public enum Errors {
    NotInitialized = -1,
    Unknown = 1,

    NoAccessToken = 1000,
    StorageRequestsMadeTooOften = 1001,
    NoSuchProduct = 1002,
    StorageError = 1003,

    UnityPurchasing_PurchasingUnavailable = 1010,
    UnityPurchasing_NoProductsAvailable = 1011,
    UnityPurchasing_AppNotKnown = 1012,
    UnityPurchasing_ProductIsNotAvailable = 1013,
    UnityPurchasing_PurchaseFailed = 1014,

    Nutaku_Error = 1100,
};
```

If everything is ok, you can read `responseData.Data` if needed.

Data Scheme

During the [Deploy](#), Balancy generates JSON files and uploads them to our CDN. For each template, we create an individual file with the version suffix.

The main file, where you can find the latest versions for all the JSON, is located here:

```
https://cdn.unnychat.com/entities/{game_id}/{environment}/versions.json
```

Parameter	Description
{game_id}	guid of your game; You can find it the dashboard
{environment}	Environment: dev, stage, production

For example, if you deployed from the DEV environment for the game with guid `11111111-1111-1111-111111111111`, then the address of your versions file is next:

```
https://cdn.unnychat.com/entities/11111111-1111-1111-111111111111/dev/versions.json
```

Versions File Structure

```
{
  "dictionaries": [
    {
      "version": "3",
      "name": "EnemyInfo",
      "id": "10587"
    },
    {
      "version": "5",
      "name": "DamageInfo",
      "id": "12"
    }
  ]
}
```

The key `dictionaries` value contains the list of all the templates' info.

Smart Objects

Smart Objects help you create and work with player's progress. It's built on the top of the Storage, but has a lot of automations. For example it generates the code required to work with the profile, it automatically tracks the changes and saves them in the cloud. If there is no internet connection, profile will be saved locally, and synchronized with the cloud was the connection is restored.

Storage

Allows you to Save and Load any data on Balancy servers. Ensure to authorize the player (at least as a guest) before using the Storage because all records are connected to specific players.

General Information

Collections and Keys

Each player can have a set of Collections, and each collection can have a set of Keys. When you save to the Storage, you must specify a Collection and a key. However, to Load the data, there are two options:

1. Load a specific key from a specific collection
2. Load the whole collection.

You can think of a Collection as a Book, where a Key is a book chapter. You can have as many books as you want with many different chapters in each one. It's up to you what you want to write in each chapter.

Both Collection and Key are string values. Ensure you are loading from the Collection/Key pair you used to save your data.

For example

You might have a collection '**Player**' with many keys: 'Inventory', 'Spells', 'Stats', etc... When one of the data changes, you need to update only a small portion, which will be stored in one key. But when you start a game, you can load the whole profile with all the keys

Versions

Another worth mentioning topic is **versions**. Balancy handles them automatically, but it would be better for you to understand how it works. Each record in the database has a version number, which increases every time you change the data. It's used to prevent using outdated data.

FAQ

How to Accelerate Balancy Launch Time?

Quick app launches are essential for a positive user experience, as delays can be off-putting. Balancy offers a solution for apps that can be launched offline, minimizing the impact of web requests on the initial user experience.

1. Pre-download Data:

- Prioritize [downloading data](#) before building your app.

2. Initialization with PreInit:

- During [initialization](#) of Balancy, use the parameter `PreInit = PreInitType.LoadStaticDataAndLocalProfile`. This approach ensures the app launches using locally available data.

3. Handling OnInitProgress Callback:

- Monitor for two additional cases in the `OnInitProgress` callback: `BalancyInitStatus.PreInitFromResourcesOrCache` and `BalancyInitStatus.PreInitLocalProfile`. Full usage of Balancy functionalities is advisable after the latter status.

4. Background Initialization:

- Balancy continues to initialize in the background. Upon completion of online data and profile synchronization, the `OnReadyCallback` will notify you.

5. Conflicts Resolving:

- Make sure you are resolving conflicts using the method `OnSystemProfileConflictAppeared()`, most likely you need to choose the Cloud version: `Balancy.LiveOps.Profile.SolveConflict(ConflictsManager.VersionType.Cloud);`.

6. Immediate Functionality Post-PreInit:

- Most features, except A/B Tests, become operational immediately after `BalancyInitStatus.PreInitLocalProfile`. To enable A/B Tests without awaiting full Balancy initialization, set the A/B Test parameter `PreInitLaunch` to `true`. Bear in mind, you won't be able to change any A/B Test parameters on the fly with this flag. You can only stop such a test or update it with the next app release.

7. Final Initialization:

- Once all new updates from Balancy are downloaded and the cloud profile is loaded and synchronized with the local progress, `AppConfig.OnReadyCallback` and `void OnSmartObjectsInitialized();` are invoked.

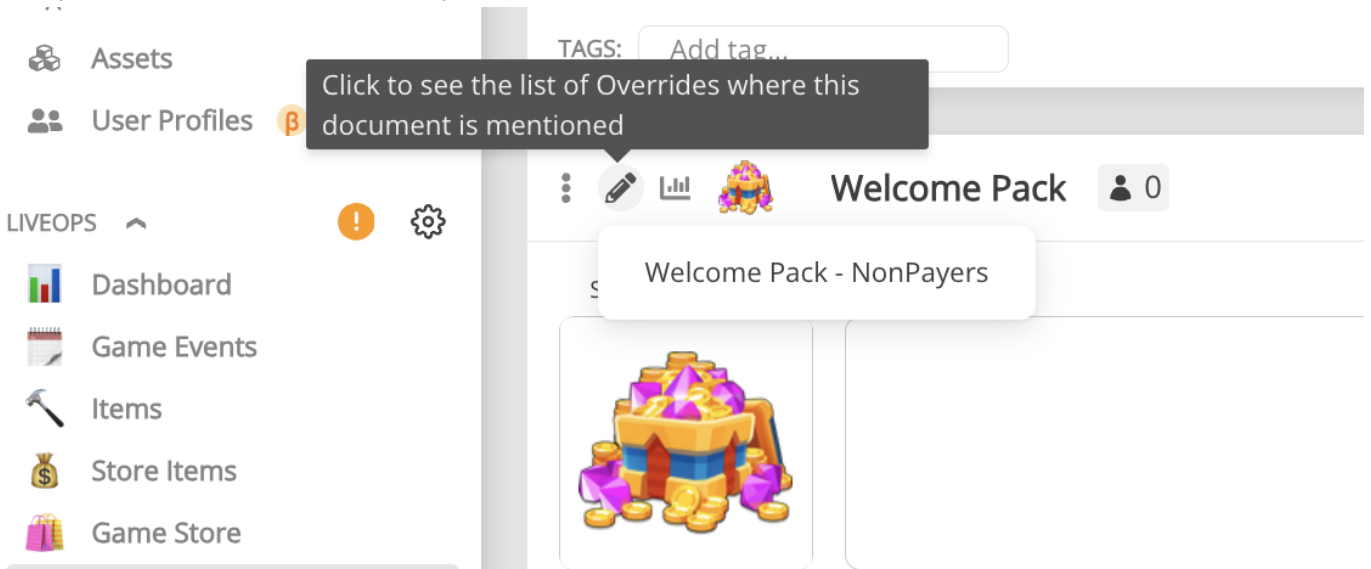
Release Notes

November 5, 2024 [Web]

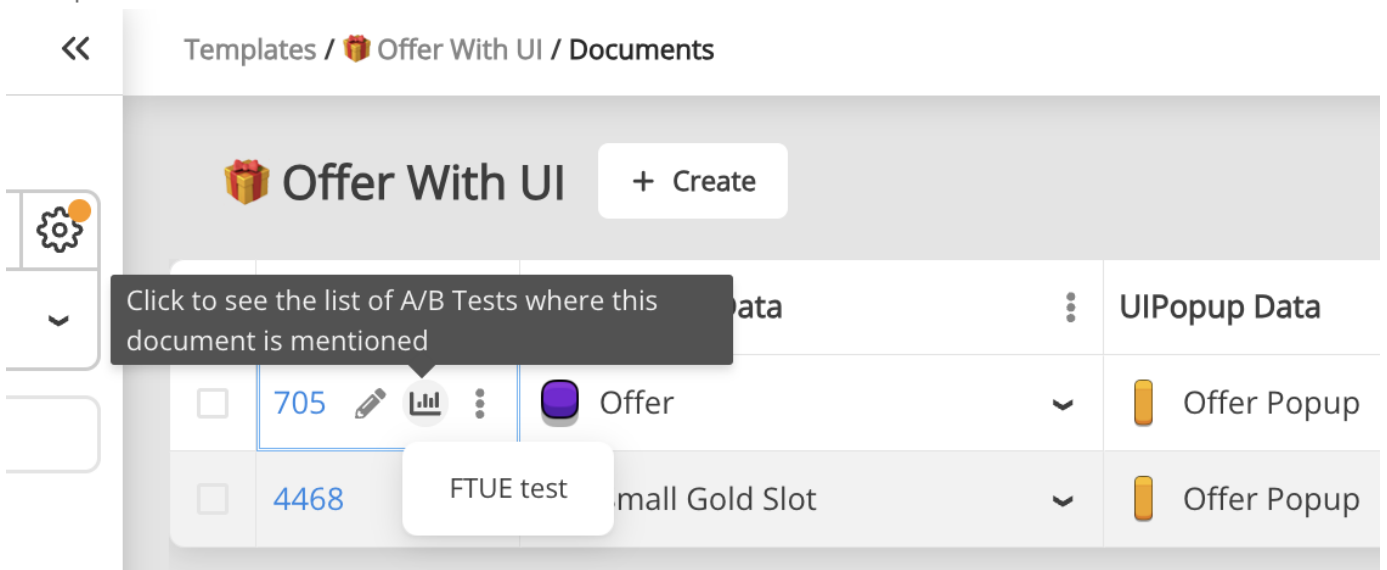
Web

- For the documents, which are subjects to overriding or A/B testing, there are now hints in Table View and LiveOps view. Clicking on the hint provides the list of overrides or A/B tests involved, for easy referencing and checking their settings. Items in the lists are clickable as well, allowing to open corresponding document.

Example: Hint about override in LiveOps view for an offer:



Example: Hint about A/B test in Table View for an offer:



Migration to branches

The main difference between the new version of Balancy platform and the old one – [Branches](#). On top level it reminds branches in systems like Git. It helps with collaborative work and versioning.

Previously users had only 3 [environments](#) - development, stage and production, and it was hard to:

1. Work simultaneously on different features.
2. Make partial rollouts.
3. Release several builds with different game balances.
4. etc

The new system with branches solves all these issues.

1. Several teammates could work in different branches on different features without interfering with each other.
2. You can release different builds with different versions on game balance.
3. During the merge process we move only changes. Previously Balancy performed only complete migration, replacing the content of target environment.

Key Changes

Migrating from the Old System

1. 3 environments will be transformed into branches with corresponding names.
2. You will have to choose which one of them will become the **main** branch.

Choose branch

Branches system is based on commits history, old system didn't have it. So you have to choose the main branch for your old projects, which you start working with. You will still be able to change data in 2 other branches and deploy them, but will not be able to create new branches from them or merge from/to them. That means:

1. You will have to make feature-freeze on those two other branches, making no development there, OR
2. Update your game before Branches are released, so you could select Dev branch as initial branch, OR
3. Postpone game update until Branches are released and make all process starting from the initial branch using new system.