AF Inventory System

User Manual & API Reference

Version 1.1
Unity uGUI • ArcanaFutura.Inventory

Contents

1. Overview
2. Package Concepts
3. Included: AF Wrap Layout Group
4. Genre-Agnostic Design
5. Quick Start
6. AF Inventory Database
7. AFInventory Component
8. AFInventoryView Settings
9. Runtime Interaction
10. Capacity, Stacks and Overflow
11. Item Combination
12. Visual Customization
13. Public API — AFInventory
14. Public API — AFInventory Events
15. Public API — Database and Definitions
16. Public API — AFInventoryView
17. API Examples
18. Integration Guidance
19. Troubleshooting
20. Demo Content and AI-Generated Graphics
21. Final Checklist
22. Contact
23. License and Disclaimer

1. Overview

AF Inventory System is a project-agnostic Unity uGUI inventory solution built around a ScriptableObject database, runtime inventory component, configurable inventory view, item prefabs, recipes, and an Examine panel.

The system deliberately separates inventory data and behavior from presentation. The same runtime inventory can therefore be presented as a plain utility bar, a heavily themed horror interface, a vertical science-fiction inventory, or a completely custom UI without rewriting the inventory logic.

Core capabilities:

- Horizontal and vertical inventory presentation.
- AF Wrap Layout Group included with the package for automatic multi-row / multi-column item layout.
- ScriptableObject database containing categories, item definitions, and combination recipes.
- Stackable and non-stackable items.
- Optional finite inventory capacity.
- Per-inventory and per-item overflow behavior.
- Starting items.
- Selection and deselection.
- Item quantities.
- Item combination through recipes.
- Examine panel support.
- Configurable inventory icon and close controls.
- Optional Always Visible mode for persistent inventory/tool/action bars, with optional manual hiding.
- Optional screen-edge activation with configurable activation/deactivation distances and selected-item hold-open behavior.
- Configurable item backgrounds, frames, icon backgrounds, icon frames, margins, selection color, quantity position, and item-name visibility.
- Per-item visual background override.
- Runtime events and a public C# API.
- Automatic Previous / Next navigation state.
- Themed presentation without coupling game logic to the supplied graphics.

Namespace:

ArcanaFutura.Inventory

The supplied system uses Unity uGUI. AF Wrap Layout Group is used internally by the inventory view/layout implementation supplied with the package.

2. Package Concepts

`AFInventoryDatabase`

The central ScriptableObject data source. It stores Categories, Items, and Recipes and provides runtime lookup methods.

`AFInventory`

The runtime inventory model. It owns the current item instances, quantities, capacity rules, selection state, overflow handling, combination logic, and inventory events.

`AFInventoryView`

The uGUI presentation/controller for an AFInventory. It creates and refreshes item views, controls opening and closing, navigation, item selection presentation, Examine behavior, and visual configuration.

`AFInventoryItemDefinition`

The database definition of an item: ID, display name, icon, optional background, description, developer notes, category, tags, action flags, slot behavior, overflow behavior, and stack settings.

`AFInventoryItem`

A runtime inventory entry. It references an AFInventoryItemDefinition and carries its current quantity.

`AFInventoryRecipe`

Defines a combination of two item definitions and its resulting item. Recipes are unordered: A + B is treated as the same pair as B + A.

`AFInventoryItemView`

The runtime visual representation of an inventory item.

`AFInventoryExaminePanel`

Displays an item's name, optional icon, and description in the Examine UI.

`AFInventoryCategory`

A simple database category identified by ID and display name.

`AFInventoryStartingItem`

Defines an item ID and starting quantity to add when the inventory initializes.

AF Wrap Layout Group

Included with AF Inventory System. This reusable Arcana Futura uGUI layout component arranges inventory items automatically across multiple rows or columns and is used by the supplied inventory prefabs. No separate purchase or installation is required. It may also be used independently elsewhere in your project.

3. Included: AF Wrap Layout Group

AF Inventory System includes AF Wrap Layout Group, Arcana Futura's reusable uGUI wrapping layout component.

It is used by the supplied inventory prefabs to arrange items automatically across multiple rows or columns and to support the horizontal and vertical inventory layouts.

AF Wrap Layout Group is included directly with AF Inventory System:

- No separate purchase is required.
- No separate installation is required.
- The included component may also be used independently elsewhere in your project.

This is a bundled component, not an external third-party dependency.

4. Genre-Agnostic Design

AF Inventory System is genre-agnostic. It is not tied to RPG conventions, a specific character controller, equipment model, combat system, quest framework, or visual style.

It is suitable for projects including:

- Adventure and point-and-click games
- First-person and third-person games
- Shooters
- Horror and survival horror
- Science-fiction games
- Puzzle and escape-room games
- RPGs and action RPGs
- Exploration and narrative games
- Survival and crafting games
- Simulations and other inventory-driven projects

The item action flags and public API describe what an item can do without prescribing how your game must implement it. A keycard, weapon, clue, potion, tool, quest object, alien egg, or potted plant can all use the same inventory core while the project supplies its own gameplay behavior.

The supplied demos intentionally use very different layouts and visual styles to demonstrate this separation between inventory functionality and presentation.

5. Quick Start

1. Create or use an AF Inventory Database asset.

   Use the Arcana Futura / AF Inventory System / Inventory Database creation menu.

2. Open the database and create the categories and items required by your project.
3. Give every item a unique ID. Runtime string lookups and recipes use these IDs.

4. Configure item behavior:
   - whether it can be dropped, given, used, opened, examined, or combined;
   - **whether it is a quest item;**
   - **whether it occupies an inventory slot;**
   - **whether it is stackable and its maximum stack size;**
   - optional per-item overflow behavior.
5. Create recipes for any item combinations.
6. Place one of the supplied inventory prefabs in a Canvas, or create your own from the supplied structure.
7. Assign the database to AFInventory.
8. Configure Max Slots, Overflow Mode, and Starting Items on AFInventory.
9. Configure AFInventoryView:
   - **orientation;**
   - **item prefab and content;**
   - **visible rows;**
   - **navigation;**
   - **open/close behavior;**
   - **inventory icon;**
   - **close button;**
   - **item visuals;**
   - Examine panel.
10. Enter Play Mode and verify:
    - **starting items appear;**
    - **selection works;**
    - **navigation buttons enable/disable correctly;**
    - **Examine works for examinable items;**
    - **configured recipes combine correctly;**
    - capacity and overflow behavior match the intended design.

The supplied demo scenes are useful references for both conventional and heavily customized presentation.

6. AF Inventory Database

The database is the authoring source for inventory content. It contains three main groups: Categories, Items, and Recipes.

4.1 Categories

Each category contains:

- ID — stable identifier used by item definitions.
- Category Name — human-readable category name.

Categories are optional organizational data. An item stores the category ID rather than a direct category object reference.

4.2 Item definitions

Each item definition contains:

ID

A unique runtime identifier. Keep IDs stable after integration because code, starting items, and recipes may refer to them.

Display Name

The player-facing item name.

Icon

Sprite used by the inventory item and Examine panel when enabled.

Background

Optional per-item background sprite. When item backgrounds are enabled in AFInventoryView, this can override the global default item background.

Description

Player-facing description, primarily used by Examine.

Dev Notes

Internal authoring notes. These are available from the definition but are not intended as player-facing content.

Category

The category ID assigned to the item.

Tags

Arbitrary string tags available to project code.

Action flags

- Can Drop
- Can Give
- Can Use
- Can Open
- Can Examine
- Can Combine
- Is Quest Item

These flags describe the capabilities of the item and are available through the public definition API. Can Examine and Can Combine are also used directly by the supplied inventory interaction flow.

Uses Inventory Slot

Controls whether the item counts toward Max Slots. This is useful for items such as special resources or other inventory entries that should exist in the inventory without consuming normal slot capacity.

Overflow Mode

Per-item overflow behavior. Use Inventory Default delegates to the AFInventory setting; otherwise the item can explicitly use Discard Overflow, Drop Overflow, or Reject All.

Stackable

When enabled, quantities may share stacks.

Max Stack

Maximum quantity in one stack. A value of 0 means unlimited.

4.3 Recipes

Recipes define Item A + Item B -> Result Item.

Recipes are unordered. A + B and B + A are the same combination.

A recipe can control whether Item A and Item B are consumed and the quantity of the result that is created. The runtime validates the recipe, input availability, item combination permissions, quantity requirements, and capacity before completing the combination.

A recipe that combines an item with itself requires two units to exist.

7. AFInventory Component

```
AFInventory is the runtime data/model component and should be treated as the authoritative
inventory state.
```

Database

Assign the AFInventoryDatabase used by this inventory.

Max Slots

Maximum number of occupied inventory slots.

- 0 = unlimited.
- Items with Uses Inventory Slot disabled do not count toward this limit.

- Stack quantity does not by itself mean multiple occupied slots; occupied runtime entries are what matter.

Overflow Mode

Default behavior when an Add operation cannot fit all requested items:

- Discard Overflow — add what can fit and discard the remainder.
- Drop Overflow — add what can fit and report the remainder through OverflowDropped so project code can create a world drop or other behavior.
- Reject All — reject the add operation when the complete request cannot be accepted.

An item's own Overflow Mode can override this default.

Starting Items

A list of Item ID + Quantity entries added during initialization. The IDs must exist in the assigned database. Missing IDs are ignored with a warning.

Runtime state exposed by AFInventory includes:

- Database
- Items
- SelectedItem
- MaxSlots
- OverflowMode
- UsedSlots
- AvailableSlots
- IsFull

Important: use the public AFInventory methods to change inventory state rather than modifying runtime lists directly. This preserves quantity rules, capacity checks, selection behavior, and event notifications.

8. AFInventoryView Settings

```
AFInventoryView is the supplied uGUI presentation/controller. It requires AFInventory on the same GameObject.
```

6.1 Orientation

Orientation

Select Horizontal or Vertical. The supplied view adjusts its layout/navigation behavior to the chosen orientation.

6.2 Items

Item Prefab

Prefab instantiated for runtime inventory entries.

Content

RectTransform that receives generated item views.

Background

Reference to the inventory background controller used by the supplied prefab structure.

Min Visible Rows

Minimum visible row/column count used by automatic sizing.

Max Visible Rows

Maximum visible row/column count before navigation/scrolling is required.

Max Slots

The custom inspector also exposes the AFInventory Max Slots setting so capacity can be configured while working on the view.

Selected Color

Tint used by the selection overlay of a selected item.

6.3 Item Visuals

Show Item Name

Globally controls whether item names are displayed in inventory item views. Disabling the name also allows the item's layout to reclaim the name area when the prefab is configured accordingly.

Background / Frame Prefab

Generic visual-layer prefab used to create runtime item background/frame layers and reused by Examine framing.

Use Item Background

Enables item backgrounds.

Default Item Background

Global background used when an item does not supply its own Background override.

Item Background Margins

Offsets applied to the generated item background layer.

Use Item Frame

Enables the global item frame.

Default Item Frame

Sprite used as the item frame.

Item Frame Margins

Offsets applied to the generated item frame.

Use Icon Background

Enables a background specifically behind the item icon.

Default Icon Background

Sprite used behind item icons.

Icon Background Margins

Offsets for the icon background.

Use Icon Frame

Enables a frame around the icon area.

Default Icon Frame

Sprite used as the icon frame.

Icon Frame Margins

Offsets for the icon frame.

Icon Margins

Insets applied to the actual item icon.

Quantity Position

Selects the corner used for quantity text: Top Left, Top Right, Bottom Left, or Bottom Right.

Item background precedence

The intended behavior is:

11. Use Item Background ON + global background assigned + no item override -> global background is shown.
12. Use Item Background ON + global background assigned + item Background assigned -> item background overrides the global background.
13. Use Item Background ON + no global background + item Background assigned -> the item background is shown.
14. Use Item Background OFF -> no item background is shown.

Frames currently use the configured global frame; item definitions provide the individual Background override.

6.4 Navigation

Scroll Rect

The ScrollRect used by the inventory.

Previous Button / Next Button

Buttons used to move through the inventory. Their Interactable state is updated automatically according to available content.

Styling note:

If a Button is placed on a transparent parent panel and the visible arrow is a child Image, set the Button's Target Graphic to the visible arrow Image. Unity Color Tint only affects the assigned Target Graphic.

6.5 Open / Close

Start Open

Controls whether a normal inventory bar begins open. This setting is not used when Always Visible is enabled.

Always Visible

Keeps the inventory open by default and prevents normal closing. This mode is intended for persistent inventory types such as weapon, flashlight, tool, or action bars.

Allow Manual Hide

Available when Always Visible is enabled. Allows the persistent inventory to be hidden and shown manually through the normal inventory toggle controls. When disabled, Close requests are ignored.

Open On Screen Edge

Available for normal inventories when Always Visible is disabled. Opens the inventory when the pointer approaches the relevant screen edge. The relevant edge is determined automatically from the inventory orientation and its actual screen position: horizontal inventories use the nearer top/bottom edge, while vertical inventories use the nearer left/right edge.

Activation Distance

Distance in pixels from the detected screen edge at which the inventory opens.

Close On Distance

When enabled with Open On Screen Edge, automatically closes the inventory after the pointer moves away from the open inventory.

Deactivation Distance

Distance in pixels from the actual open inventory bounds beyond which Close On Distance closes the inventory.

Keep Open If Item Selected

When enabled, Close On Distance is ignored while an inventory item is selected. Once no item is selected, the normal deactivation-distance rule applies again.

Inventory Bar

The RectTransform shown/hidden by the view.

6.6 Inventory Icon

Show Icon

Enables the inventory toggle icon/button.

Show Icon When Open

Controls whether the inventory icon remains visible while the inventory itself is open.

Inventory Icon Button Prefab

Prefab used to create the inventory icon button.

Icon Sizing

Controls how the inventory icon is sized. Fit To Bar uses the bar-oriented sizing behavior; Custom uses the prefab's RectTransform dimensions.

Button Placement

Controls whether the button is embedded in the inventory layout or placed independently according to the supplied placement behavior.

Icon Position

Side/position used for the inventory icon.

Icon Alignment

Alignment along the relevant edge.

Icon Space

Controls whether the icon occupies inside/outside space relative to the bar.

For Custom icon sizing, duplicate the supplied icon prefab before changing its RectTransform if you want to preserve the original.

6.7 Close Button

Show Close Button

Enables the inventory Close button.

Close Button

Button used to close the inventory.

Close Button Position

Corner used to anchor/pivot the button.

The position setting changes the anchor/pivot but preserves the prefab's configured Pos X / Pos Y offset. This allows the prefab designer to determine the exact distance from decorative borders.

6.8 Examine

Examine Panel Prefab

Prefab instantiated for item examination.

Examine Close Button Position

Selects whether the Examine close button is positioned on the left or right.

Right Click Same Item to Close Examine

When enabled, right-clicking the item that is already being examined closes the Examine panel.

Show Item Icon

Controls whether the Examine panel displays the item's icon. When the supplied Icon Panel reference is configured, disabling the icon also collapses its panel/layout space.

Use Background

Enables the Examine background.

Background

Sprite applied to the existing Examine Content Panel background Image.

Use Frame

Enables the Examine frame.

Frame

Sprite instantiated using the shared Background / Frame Prefab. The frame is created as the final child so it renders above the Examine content.

Examine visuals intentionally remain simpler than item visuals: there is one Examine surface, so there is no per-item frame/background hierarchy or separate margin system.

9. Runtime Interaction

The supplied interaction flow is intentionally straightforward and can be integrated with project-specific gameplay systems.

Selection

Clicking an inventory item selects it. Selection is reflected by the configured Selected Color. Selecting another item changes the selection; DeselectItem can also be called through the API.

Examine

Right-clicking an item requests Examine. Examine only opens when the item's Can Examine flag is enabled. If Right Click Same Item to Close Examine is enabled, right-clicking the currently examined item closes the panel.

Combination

The supplied view supports combination interaction using the selected item and a target item. The current item view exposes middle-click interaction for this flow; AFInventory then validates and attempts the recipe through TryCombine.

Projects can also call TryCombine directly and build their own context-menu, controller, touch, or other interaction layer around the public inventory API.

Item action flags

Can Drop, Can Give, Can Use, Can Open, Can Examine, Can Combine, and Is Quest Item are exposed as definition data. AF Inventory System does not impose game-specific implementations for actions such as Use, Give, Open, or Drop. Your game decides what those actions mean.

This separation is intentional: an inventory item can advertise its capabilities without forcing AF Inventory System to know about doors, characters, weapons, quests, world pickups, or other project-specific systems.

10. Capacity, Stacks and Overflow

Unlimited inventory

Set Max Slots to 0.

Finite inventory

Set Max Slots above 0. UsedSlots counts runtime inventory entries whose definitions have Uses Inventory Slot enabled.

Stackable items

A stackable definition can accumulate quantity according to Max Stack.

- Max Stack = 0 means unlimited per stack.
- A finite Max Stack can cause additional runtime stacks to be created when quantity exceeds one stack.

Non-stackable items

Quantities are represented as separate runtime item instances rather than one stack.

CanAddItem

Use this before adding when your game needs a simple capacity/validity check.

AddItem

Returns AFInventoryAddResult containing:

- Requested
- Added
- Overflow
- Complete
- HasOverflow

Overflow behavior

The effective behavior is determined by the item override when present, otherwise by AFInventory. Overflow events allow project code to respond to dropped or rejected quantities.

Items that do not use inventory slots

Uses Inventory Slot can be disabled for definitions that should be tracked without consuming normal capacity.

11. Item Combination

Combination is database-driven.

Before a combination succeeds, AFInventory validates:

- both runtime items are valid;
- both belong to this inventory;
- both definitions allow combination;
- a recipe exists;
- required quantities exist;
- the recipe result can fit under current capacity rules.

TryCombine returns AFInventoryCombineResult:

- Success
- InvalidItem
- ItemNotInInventory
- CombinationNotAllowed

- RecipeNotFound
- InsufficientQuantity
- InsufficientCapacity

On success:

- configured recipe inputs are consumed;
- the result quantity is added;
- the current selection is cleared;
- ItemsCombined is raised.

On failure:

- no successful combination is performed;
- CombinationFailed is raised with the failure result.

Recipes are unordered, so Item A + Item B is equivalent to Item B + Item A.

12. Visual Customization

AF Inventory System is designed so functionality does not depend on the supplied visual style.

Recommended workflow:

15. Duplicate supplied prefabs before making theme-specific changes.
16. Keep functional references assigned.
17. Replace sprites, colors, fonts, spacing, and sizing as required.
18. Use AFInventoryView settings for common visual layers rather than hard-wiring artwork into every item prefab.
19. Use the item definition Background only when a particular item needs to override the global item background.
20. Keep Button Target Graphic assignments pointed at the visible child graphic when the clickable parent itself is transparent.
21. For close buttons, set the desired Pos X / Pos Y on the prefab; corner selection preserves those values.
22. Use sliced sprites where the frame/background needs to resize. Observe Unity texture compression requirements for the selected import format.

The supplied themed demos demonstrate that the same runtime system can support very different layouts and art directions. Demo artwork is presentation content only and is not required by the runtime inventory system.

13. Public API — AFInventory

Properties

```
AFInventoryDatabase Database
```

The assigned database.

IReadOnlyList<AFInventoryItem> Items

Current runtime inventory entries.

```
AFInventoryItem SelectedItem
```

Currently selected runtime item, or null.

int MaxSlots

Configured slot limit. 0 means unlimited.

```
AFInventoryOverflowMode OverflowMode
```

Default inventory overflow mode.

int UsedSlots

Number of occupied slot-using runtime entries.

int AvailableSlots

Remaining capacity, or int.MaxValue for unlimited inventories.

bool IsFull

True when a finite inventory has no remaining slot capacity.

Queries

bool IsInInventory(AFInventoryItemDefinition definition)

bool IsInInventory(string id)

Returns whether at least one matching item is present.

bool IsSelected(AFInventoryItemDefinition definition)

bool IsSelected(string id)

Returns whether the current selection matches the supplied definition/ID.

```
AFInventoryItem GetItem(AFInventoryItemDefinition definition)
AFInventoryItem GetItem(string id)
```

Returns a matching runtime item or null.

IReadOnlyList<AFInventoryItem> GetItems(AFInventoryItemDefinition definition)

Returns all runtime entries for a definition.

int GetQuantity(AFInventoryItemDefinition definition)

int GetQuantity(string id)

Returns the total quantity across matching runtime entries/stacks.

Adding

```
AFInventoryAddResult AddItem(AFInventoryItemDefinition definition, int quantity = 1)
```

Attempts to add quantity and returns requested/added/overflow information.

bool CanAddItem(...)

Checks whether the requested add can be accepted according to the relevant rules. See IntelliSense/source signature for the overload supplied by the package version.

Removing

bool CanRemoveItem(AFInventoryItemDefinition definition, int quantity = 1)

Checks whether the requested quantity can be removed.

bool RemoveItem(AFInventoryItemDefinition definition, int quantity = 1)

Removes quantity and reports success/failure.

Combination

```
AFInventoryRecipe GetRecipe(AFInventoryItemDefinition first, AFInventoryItemDefinition second)
```

Returns the matching recipe or null.

bool HasRecipe(AFInventoryItemDefinition first, AFInventoryItemDefinition second)

Returns whether a recipe exists.

bool CanCombine(AFInventoryItem first, AFInventoryItem second)

Returns true only when the combination currently passes runtime validation.

```
AFInventoryCombineResult TryCombine(AFInventoryItem first, AFInventoryItem second)
```

Attempts the combination and returns the detailed result enum.

Selection

bool SelectItem(AFInventoryItem item)

Selects a runtime item when valid.

bool SelectItem(...)

An additional selection overload is supplied for definition-based selection. Use IntelliSense/source for the exact signature in the installed package version.

void DeselectItem()

Clears the current selection.

void Clear()

Clears inventory contents through the inventory's normal state-management path.

14. Public API — AFInventory Events

ItemAdded

Action<AFInventoryItem>

Raised when a runtime item entry is added.

ItemRemoved

Action<AFInventoryItem>

Raised when a runtime item entry is removed.

ItemChanged

Action<AFInventoryItem>

Raised when an existing runtime entry changes, such as quantity changes.

ItemSelected

Action<AFInventoryItem>

Raised when an item becomes selected.

ItemDeselected

Action<AFInventoryItem>

Raised when an item is deselected.

OverflowDropped

Action<AFInventoryItemDefinition, int>

Raised when overflow uses Drop Overflow. The inventory reports the definition and overflow quantity; your game decides how to represent the dropped result in the world.

AddRejected

Action<AFInventoryItemDefinition, int>

Raised when an add is rejected according to the active overflow rules.

ItemsCombined

Action<AFInventoryItem, AFInventoryItem, AFInventoryRecipe>

Raised after a successful recipe combination.

CombinationFailed

Action<AFInventoryItem, AFInventoryItem, AFInventoryCombineResult>

Raised when TryCombine fails and includes the reason.

15. Public API — Database and Definitions

```
AFInventoryDatabase
```

IReadOnlyList<AFInventoryCategory> Categories

IReadOnlyList<AFInventoryItemDefinition> Items

IReadOnlyList<AFInventoryRecipe> Recipes

```
AFInventoryItemDefinition GetItem(string id)
```

Looks up an item by ID.

```
AFInventoryCategory GetCategory(string id)
```

Looks up a category by ID.

```
AFInventoryCategory GetCategory(AFInventoryItemDefinition item)
```

Returns the category assigned to an item.

```
AFInventoryRecipe GetRecipe(AFInventoryItemDefinition first, AFInventoryItemDefinition second)
AFInventoryRecipe GetRecipe(string firstId, string secondId)
```

Returns the unordered pair recipe.

bool HasRecipe(AFInventoryItemDefinition first, AFInventoryItemDefinition second)

bool HasRecipe(string firstId, string secondId)

Checks for a recipe.

```
AFInventoryItemDefinition GetRecipeItemA(AFInventoryRecipe recipe)
AFInventoryItemDefinition GetRecipeItemB(AFInventoryRecipe recipe)
AFInventoryItemDefinition GetRecipeResult(AFInventoryRecipe recipe)
```

Resolve recipe IDs back to database definitions.

```
AFInventoryItemDefinition read-only properties
```

string Id

string Name

Sprite Icon

Sprite Background

string Description

string DevNotes

string CategoryId

IReadOnlyList<string> Tags

bool CanDrop

bool CanGive

bool CanUse

bool CanOpen

bool CanExamine

bool CanCombine

bool IsQuestItem

bool UsesInventorySlot

```
AFInventoryItemOverflowMode OverflowMode
```

bool Stackable

int MaxStack

```
AFInventoryItem read-only properties
AFInventoryItemDefinition Definition
```

string Id

int Quantity

16. Public API — AFInventoryView

Properties

```
AFInventory Inventory
```

The AFInventory controlled by this view.

bool IsOpen

Current open/closed state.

Events

Opened

Raised when the inventory view opens.

Closed

Raised when the inventory view closes.

Methods

void Refresh()

Refreshes the displayed inventory state.

void Open()

Opens the inventory view.

void Close()

Closes the inventory view.

void Toggle()

Toggles between open and closed.

void Previous()

Moves navigation toward the previous inventory position/page according to the configured orientation/layout.

void Next()

Moves navigation toward the next inventory position/page according to the configured orientation/layout.

17. API Examples

Get a definition and add it

```
AFInventoryItemDefinition key =
    inventory.Database.GetItem("key");
AFInventoryAddResult result =
    inventory.AddItem(key, 1);
if (!result.Complete)
{
    Debug.Log(
        $"Added {result.Added}; overflow {result.Overflow}.");
}
```

Check possession and quantity

```
if (inventory.IsInInventory("medkit"))
{
    int quantity =
        inventory.GetQuantity("medkit");
    Debug.Log(
        $"Player has {quantity} medkit(s).");
}
```

Remove an item

```
AFInventoryItemDefinition keycard =
    inventory.Database.GetItem("keycard");
if (inventory.CanRemoveItem(keycard, 1))
    inventory.RemoveItem(keycard, 1);
```

Listen for inventory changes

```
private void OnEnable()
{
    inventory.ItemAdded += OnItemAdded;
    inventory.ItemRemoved += OnItemRemoved;
    inventory.ItemChanged += OnItemChanged;
}
private void OnDisable()
{
    inventory.ItemAdded -= OnItemAdded;
    inventory.ItemRemoved -= OnItemRemoved;
    inventory.ItemChanged -= OnItemChanged;
}
private void OnItemAdded(AFInventoryItem item)
{
```

```
        Debug.Log(
            $"Added: {item.Definition.Name}");
    }
```

React to dropped overflow

```
    private void OnEnable()
{
        inventory.OverflowDropped += OnOverflowDropped;
    }
    private void OnDisable()
{
        inventory.OverflowDropped -= OnOverflowDropped;
    }
    private void OnOverflowDropped(
        AFInventoryItemDefinition definition,
        int quantity)
{
        // Spawn a world pickup, send to another container,
        // or apply project-specific handling here.
    }
```

Combine two runtime items

```
    AFInventoryCombineResult result =
        inventory.TryCombine(
            firstItem,
            secondItem);
    if (result != AFInventoryCombineResult.Success)
{
        Debug.Log(
            $"Combination failed: {result}");
    }
```

Control the inventory UI

inventoryView.Open();

inventoryView.Close();

inventoryView.Toggle();

The examples intentionally use the public API rather than reaching into runtime collections or generated item views.

18. Integration Guidance

Use AFInventory as the model

Gameplay systems should generally communicate with AFInventory and the database, not with generated UI item objects.

Treat item IDs as stable keys

Changing an ID after it is referenced by starting items, recipes, save data, or project code can break those references.

Keep project-specific behavior outside the inventory

Can Use does not mean AF Inventory System should know how your weapon, potion, terminal, or quest object works. Subscribe to inventory/game interactions or query the item definition and execute your own gameplay behavior.

Save systems

AF Inventory System exposes the runtime item definitions/IDs and quantities required for integration with a save system, but does not force a particular save package. Save stable item IDs and quantities, then restore them through the public inventory API.

Input systems

The supplied uGUI interaction provides a working pointer-based flow using Unity's Input System. Screen-edge activation reads the current pointer position through the Input System. Projects may layer controller, keyboard, touch, context menus, drag/drop, or other input on top of the public API without changing the inventory model.

Multiple inventories

Each AFInventory instance owns its own runtime state. The same database can therefore be used by multiple inventory instances while keeping their contents separate.

19. Troubleshooting

Items do not appear

- Confirm AFInventory has a Database assigned.
- Confirm Starting Item IDs exist in that database.
- Confirm AFInventoryView has Item Prefab and Content assigned.
- Confirm the item definition exists and AddItem actually accepted it.

Icon does not appear

- Confirm the item definition has an Icon assigned.
- Confirm the Image reference exists in the item prefab.
- For Examine, confirm Show Item Icon is enabled and the Examine icon references are assigned.

Quantity exists but is not visible

- Confirm the quantity TMP object is assigned and visually enabled.
- Check its color/alpha and hierarchy.
- Verify the item quantity warrants displaying the quantity indicator.

Selection works but is not visible

- Confirm Selected Color has a visible alpha.
- Confirm the item prefab's selection Image/reference is assigned.

Background/frame does not appear

- Confirm the corresponding Use option is enabled.
- Confirm the global sprite is assigned, or for item backgrounds, confirm the item Background override is assigned.
- Confirm Background / Frame Prefab is assigned.
- Check sprite import settings and 9-slice borders where applicable.

Navigation arrow does not dim

- Confirm the Button uses Color Tint.
- Confirm Disabled Color differs from Normal.
- If the Button is on a transparent parent, set Target Graphic to the visible child arrow Image.

Examine icon hides but leaves empty space

- Confirm the Examine panel's Icon Panel reference is assigned so the whole layout panel can be disabled.

Examine frame appears behind content

- Use the runtime Examine frame option rather than a permanently embedded frame object. The generated frame is placed as the last child so it renders above the panel content.

Close button loses intended offset

- Use the current supplied scripts. Corner positioning preserves the prefab's Pos X / Pos Y values; configure the desired offset directly on the button prefab.

Recipe does not work

- Confirm both definitions have Can Combine enabled.
- Confirm a valid recipe exists.
- Confirm both items are in this inventory.
- Confirm required quantities exist.
- Confirm the result can fit in the inventory.
- For A + A recipes, ensure at least two units exist.

Add operation does not behave as expected at capacity

- Check Max Slots.
- Check Uses Inventory Slot on the item.
- Check the item's Overflow Mode.
- Check the inventory's default Overflow Mode.
- Inspect AFInventoryAddResult and relevant overflow/rejection events.

20. Demo Content and AI-Generated Graphics

The package may include demonstration graphics used to show different visual configurations of AF Inventory System. Some demonstration graphics were created with generative AI assistance.

These graphics are demonstration/example content and are not required for the functionality of AF Inventory System. The inventory runtime, database, API, editor configuration, prefab logic, and integration behavior are independent of those graphics.

Developers are expected to use, modify, or replace visual assets according to the needs of their own project.

21. Final Checklist

Before shipping your own inventory setup:

- Database assigned.
- Every item has a unique stable ID.
- Categories and tags are correct where used.
- Item action flags reflect intended gameplay.
- Stack and Max Stack settings verified.
- Uses Inventory Slot verified.
- Overflow rules verified.
- Starting Item IDs and quantities verified.
- Recipes tested in both item orders.
- Max Slots tested at capacity.
- Selection visible.
- Item name visibility set as intended.
- Quantity position and visibility checked.
- Background/frame options checked.
- Per-item background overrides checked.
- Inventory icon behavior checked open and closed.
- Always Visible and optional manual hiding checked where used.
- Screen-edge activation, activation/deactivation distances, and selected-item hold-open behavior checked where used.
- Close button corner and prefab offset checked.
- Previous/Next disabled tint checked.
- Examine background/frame/icon checked.
- Right-click Examine close behavior checked.
- Public events connected to project-specific gameplay where required.
- Custom prefabs tested at target resolutions/aspect ratios.

22. Contact

Have a question, need help, or found a bug? Please use our online contact form and we'll get back to you as soon as possible.
Website: https://arcanafutura.eu/
Support: https://arcanafutura.eu/asset-support

23. License and Disclaimer

AF Inventory System is provided under the applicable Unity Asset Store license. It may be used and modified as part of your own commercial or non-commercial projects, subject to the terms of that license.

You may modify the supplied source code for use in your projects. Modification does not grant the right to redistribute or resell AF Inventory System itself. The asset, its source code, or modified versions of the asset may not be redistributed, resold, sublicensed or otherwise made available as a standalone asset or development tool.

AF Inventory System is provided “as is”, without warranties of any kind. Arcana Futura is not liable for losses or damages resulting from its use.

AF Inventory System is provided under the Unity Asset Store End User License Agreement (EULA).
https://unity.com/legal/as-terms

Arcana Futura