VIROO Integration

Table of Contents

Introduction

This integration package enables seamless use of VR Builder, a powerful node-based authoring tool for creating interactive VR experiences, within VIROO Studio, a collaborative multi-user, multi-platform VR development framework built on Unity.

By combining the visual scripting capabilities of VR Builder with the scalable, networked environment of VIROO Studio, this package allows developers to create immersive, interactive scenarios that can be deployed across a range of devices, including desktop and VR headsets, and accessed by multiple users simultaneously.

The goal of this integration is to replicate the majority of VR Builder's core functionalities within a VIROO Studio context, preserving its modular architecture, extensibility, and intuitive workflow, while adding support for multi-user synchronization and platform interoperability. This enables teams to collaboratively author, test, and run VR experiences that are both rich in interaction and suitable for enterprise-grade deployment in training, simulation, and presentation environments.

About VR Builder

VR Builder helps you create interactive VR applications better and faster. By setting up a Unity scene for VR Builder, you will pair it with a VR Builder *process*. Through the VR Builder process, you can define a sequence of actions the user can take in the scene and the resulting consequences.

You can easily edit a process without coding through VR Builder's process editor. The process editor is a node editor where the user can arrange and connect the *steps* of the process. Each step is a different node and can include any number of *behaviors*, which make things happen in the scene. Likewise, a step will have at least one *transition* leading to another step. Every transition can list several *conditions* which have to be completed for the transition to trigger. For example, step B can be reached only after the user has grabbed the object specified in step A.

Behaviors and conditions are the "building blocks" of VR Builder. Several of them are provided in the free open-source version already. Additional behaviors and conditions are available in the commercial version, available on the Unity Asset Store and to Enterprise customers and Content Partners. Since VR Builder is open source and has an open API, you can always write your own behaviors and conditions as well.

Behaviors and conditions can interact only with *process scene objects*. These are game objects in the scene which have a `Process Scene Object` component on them.

The interaction capabilities of a process scene object can be increased by adding *scene object properties* to it. For example, adding a `Grabbable Property` component to a game object will let VR Builder know that the object is grabbable, and when it is grabbed.

Normally it is not necessary to add properties manually to an object. When an object is dragged in the inspector of a condition or behavior, the user has the option to automatically configure it with a single click.

However, in the VIROO integration, the user must then navigate to the VIROO Network Object component in the inspector and manually generate a network id for the object by clicking the appropriate button.

Where possible, properties try to add and configure required components by themselves. If you add a `Grabbable Property` to a game object, this will automatically be made grabbable in VR (it still needs to have a collider and a mesh, of course).

This makes it very easy to start from some generic assets and build a fully interactive scene.

Architecture

The VIROO Integration for VR Builder integrates the VR Builder framework with VIROO to support the development of interactive and collaborative VR applications in Unity.

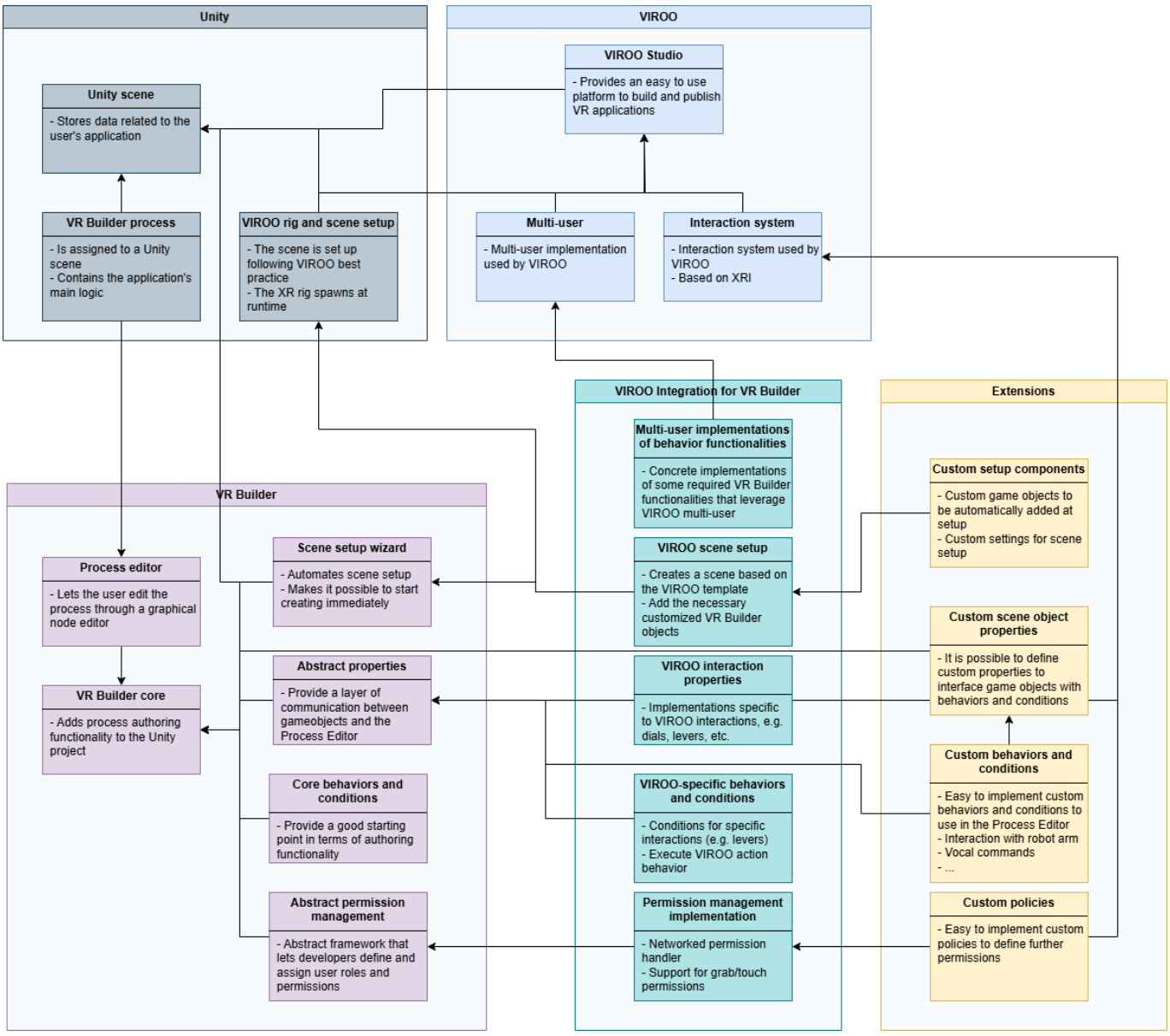
At its foundation is the Unity layer, where the Unity scene stores application data and the VR Builder process defines the main logic of the application. The VIROO rig and scene setup ensures scenes follow VIROO standards.

The VR Builder core adds authoring tools such as the Process Editor for visual logic creation and a Scene Setup Wizard to automate initial configuration. It includes core Behaviors and Conditions, abstract properties for object-editor communication, and permission management to define user roles and access levels.

The VIROO Integration module connects VR Builder to VIROO, enabling scene creation based on VIROO templates and adding custom behaviors (e.g., for levers or voice commands). It integrates VIROO's Interaction System (XRI-based), multi-user support, and permission management for interaction control.

As always, it is possible to extend this with custom behaviors and conditions, scene properties, and permission policies, offering developers flexibility to tailor the experience to their needs.

This architecture is visualized in the diagram below.



System Requirements

To ensure compatibility and optimal performance when using the VIROO Integration package with VR Builder, please verify that your development environment meets the following requirements:

Unity

- Version: Unity 2022.3 LTS (or later 2022.x releases)
- Scripting Backend: IL2CPP or Mono (IL2CPP recommended for builds)
- Rendering Pipeline: Built-in Render Pipeline (URP and HDRP are not officially supported at this time)

VR Builder

- Minimum Version: VR Builder 5.3.1
- Recommended Version: Latest stable release of VR Builder 5.x
- When importing, deselect the "Load demo scene" option and skip overwriting any existing project layers when prompted (e.g., Layer 31).

VIROO Studio

- Template Version: VIROO Studio 2.6
- Download Link: VIROO Studio Template Project
- Ensure the VIROO plugin and scene setup are properly initialized before importing the VR Builder integration.

Platforms

The integration is compatible with tethered VR devices running the VIROO Player in Windows. Mobile standalone VR (e.g., native Quest builds) is not officially supported at this time due to the PC-based architecture of VIROO Studio 2.6.

Installation

This section guides you through the steps required to set up your Unity project for integrating VR Builder with VIROO Studio. You'll begin with a VIROO Studio template project, add the necessary VR Builder components, and configure the environment to ensure compatibility between the two systems.

The installation process assumes you have Unity 2022 installed and are familiar with importing packages and navigating basic Unity project settings. By the end of this setup, you'll have a working project ready for creating collaborative, multi-user VR experiences using VR Builder within the VIROO Studio framework.

1. **Set Up VIROO Studio Template Project**
    - Unzip and open the VIROO Studio template project in Unity 2022.
2. **Import VR Builder**
    - Import a supported VR Builder version (at the time of writing, **5.3.1 or later**).
        - If a dialog asking to overwrite **Layer 31** pops up, click **"Skip"** as many times as needed.
        - On the last page of the wizard, **untick "Load demo scene"** to avoid unnecessary content.
3. **Disable VR Builder's built-in interaction system**

- Navigate to `Project Settings > VR Builder`.
- Untick **"Enable built-in interaction component"** as we'll be using the VIROO Integration instead.

4. **Import Required Packages**

   - Import the **VR Builder Permission Management package** as it is a dependency of the VIROO Integration.
   - Import the **VIROO Integration package** to complete the setup.

Setup

Create a New VIROO Scene

To begin, use the **Scene Setup Wizard** to create a new scene configured for VIROO.

1. Open Unity and navigate to `Tools > VR Builder > Scene Setup Wizard...`.
2. Select **VIROO** as the scene configuration.
3. **Recommended:** Create a *new* scene for optimal integration.
   - If you must use an existing scene, ensure it is already VIROO-compatible (the wizard will not apply VIROO-specific changes in this case).

VIROO Scene Requirements

In a VIROO scene, the following things have to be taken into account.

1. **Hierarchy Structure**

   - All game objects must be parented to the `Root` object (found in the Hierarchy).
   - VR Builder configuration objects are located under `Root > VR Builder`.

2. **Layer Configuration**

   - Assign your **floor objects** to **Layer 8 ("Physics-Floor")** to enable proper navigation and physics interactions.

3. **Network Object Setup**

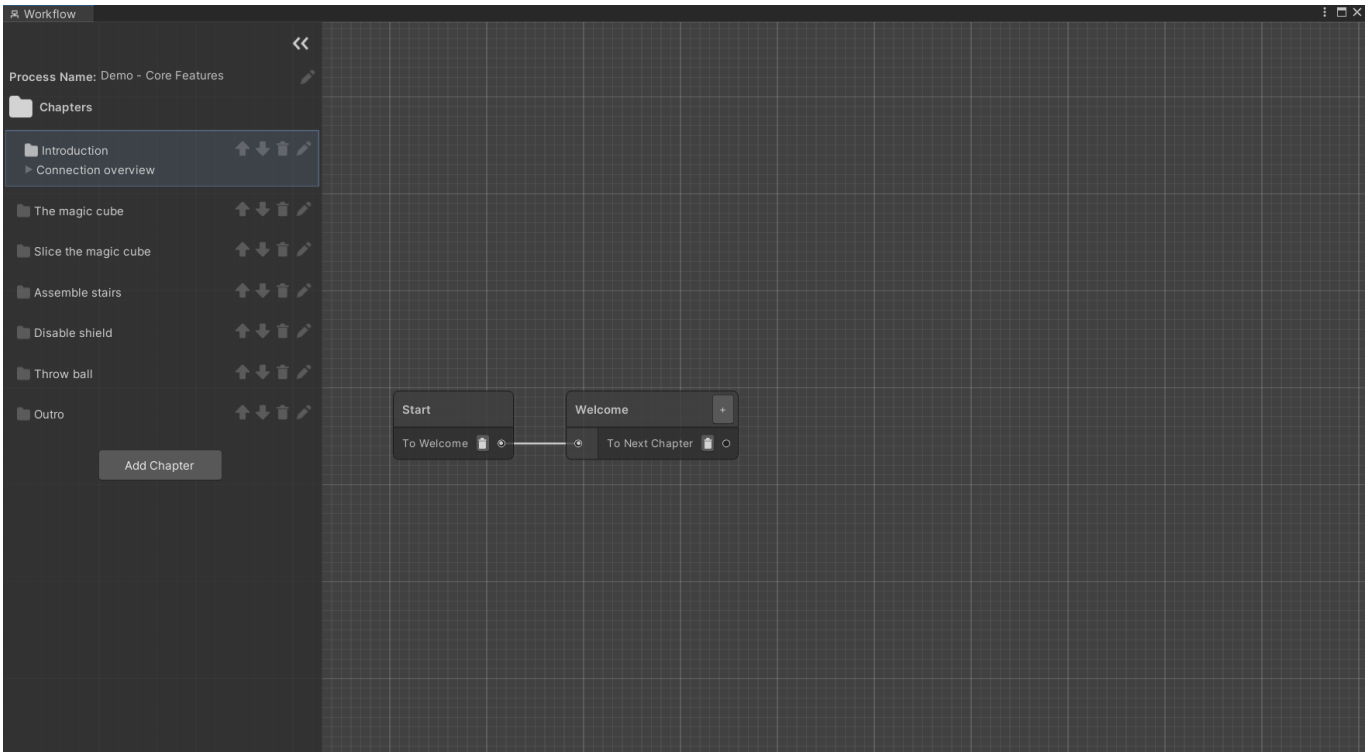
   - When configuring objects in VR Builder:
     - Use the Step Inspector to drag and configure objects as usual.
     - When a `Network Object` component is automatically added, manually click **"Generate Object ID"** to ensure proper synchronization across clients.

How to Use

It is possible to use VR Builder without restrictions within the VIROO platform. The experience and functionality is largely similar to VR Builder core with the default interaction component. Basic VR Builder functionalities are explained below for your convenience. You can reference the full VR Builder documentation at http://documentation.mindport.co.

Process Editor

The process editor lets you design the process of your VR application. You can open the process editor from `Tools > VR Builder > Process Editor` or `Window > VR Builder > Process Editor`. The process editor for the demo scene should look like this.



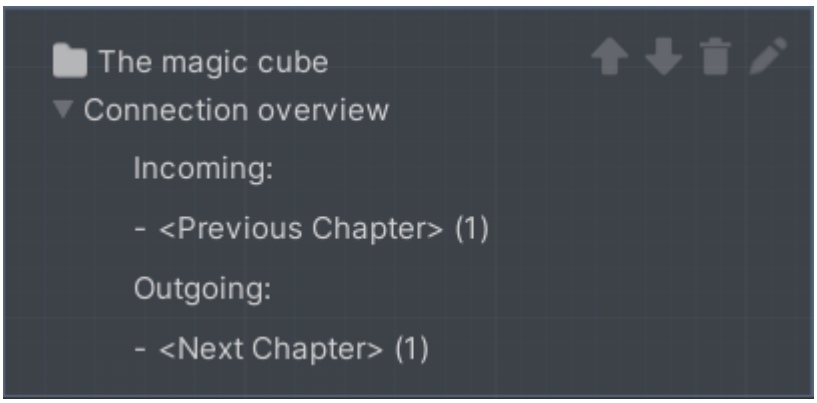
Chapters view

On the left, there is a list of chapters. Every chapter is a separate section of the process. They are useful to separate a process in its logical steps and avoid too much clutter in a single graph.

You can click on the different chapters to visualize the corresponding graphs.

Next to the chapter name, there are icons that allow you to move the chapter up and down in the list, rename it or delete it.

Underneath, you can see the `Connections breakdown` foldout. Expand it to see incoming and outgoing connections for the current chapter. That is, which chapters lead here and to which chapter it is possible to go from this one. The number next to each connection represents the amount of steps that connect to the chapter. When "Next Chapter" or "Previous Chapter" is listed as a connection, it means the connection is implicit: a path ends with an empty transition, which by default ends the current chapter and starts the next one in order.

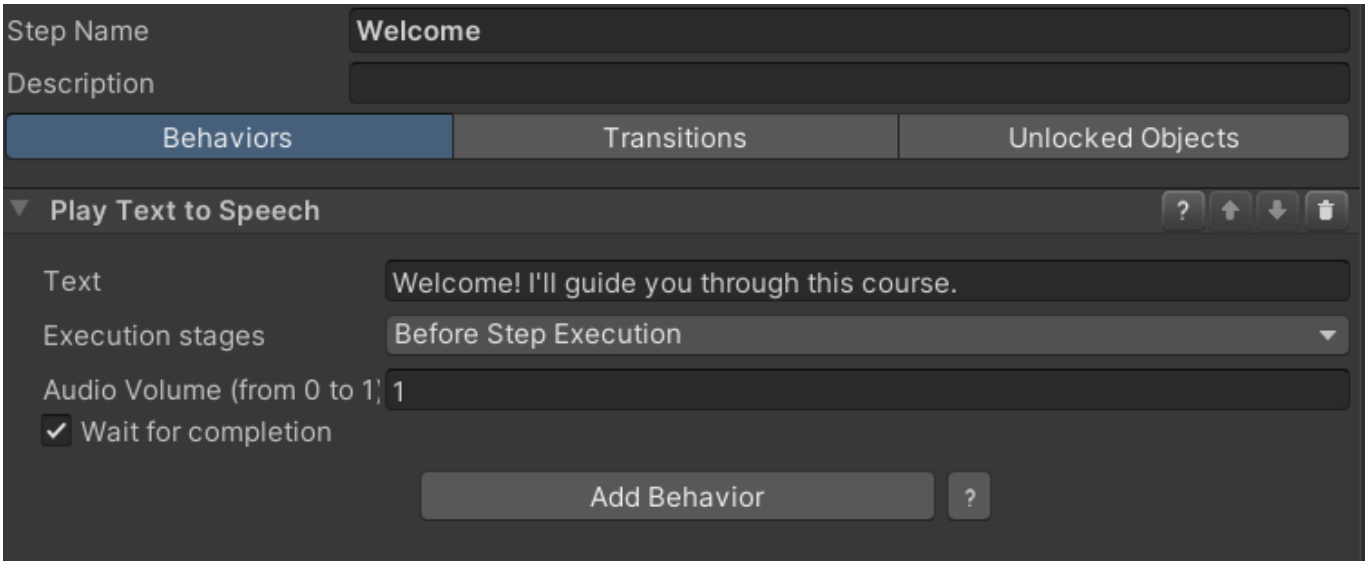


Most processes will be linear, meaning that each chapter will lead directly to the next and the connection overview only contains implicit connections, but it is possible to create more complex processes that don't follow the chapter list linearly.

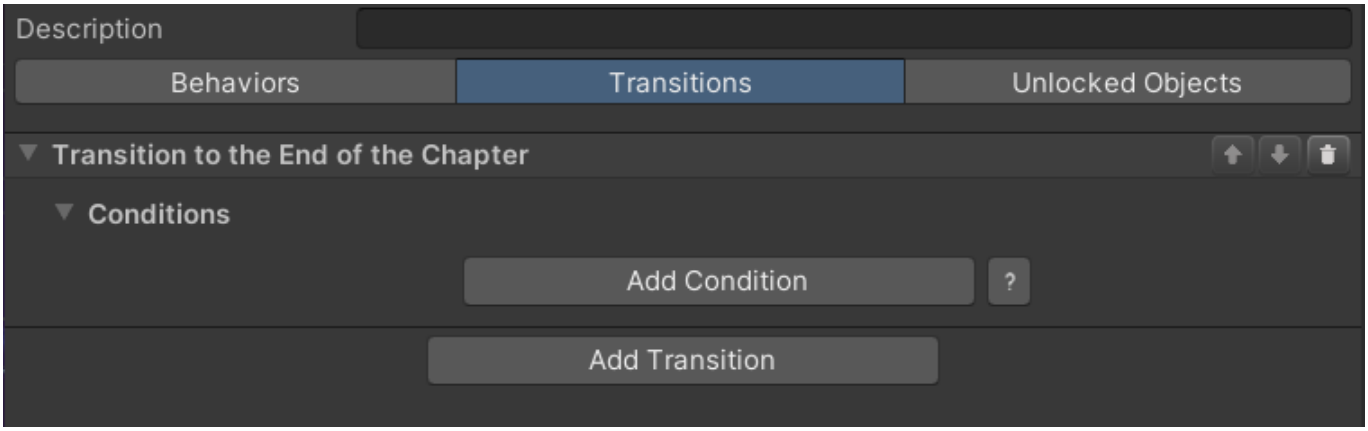
Graph view

On the right, there is a graphical representation of the current chapter. Every node is called a `Step`. Every step can include a number of `Behaviors` which can happen when the node is triggered or before leaving it. The most common uses are text to speech instructions and animations. A step can have as many exit points, called `Transitions`, as needed. Every transition can list a number of `Conditions` which, if fulfilled, make the transition valid.

Behaviors and conditions are displayed in the step inspector when a node is selected.



In the image above, the only behavior is a text to speech instruction that will be triggered when the node is entered.



This image shows a single transition. A step can have multiple transitions, each leading to a different step. In this case, the transition is connected to no other step, so it will end the chapter. The next chapter will then start.

Transitions can include conditions. If they do, they will trigger only when the attached conditions are completed. This transition has no conditions, so it will trigger immediately after the current step has ended, without any input from the user.

We encourage you to discover the behaviors and conditions available in VR Builder and the VIROO Integration.

Step Nodes

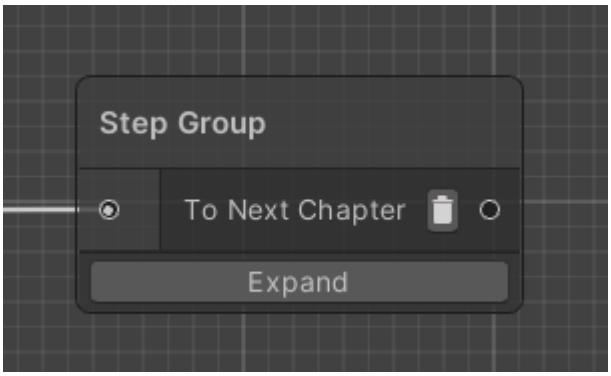
You can create a node by right clicking anywhere in the graph and selecting `New`, then the type of node you want to create. There are four types of node available in VR Builder core:

Step

This is the default step node, the main building block for your process. By default, it is empty. This means that nothing will happen, and the execution will immediately proceed to the next node, if present. You will need to add behaviors and conditions to it in the `Step Inspector` in order to customize it and build your process logic.

Step Group

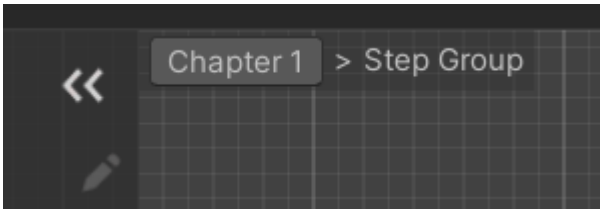
This node doesn't let you set conditions and behaviors, but instead can be expanded in a new node graph. It can be populated with other step nodes and act as a "sub-chapter" with some self contained logic. This can help keep the process tidy.



You can access the node's graph by clicking the `Expand` button or double clicking on the node itself. There are also context menu options for expanding the node or ungrouping it - that is, replacing it in the main graph with the logic it contains.

This node only has one entry and one exit point. This means that after the contained logic has ended executing, the process will always continue executing from the exit transition of the group node.

If you are in a step group graph, it will be indicated on the top left of the process editor.



You can click on a parent to return to it. For example, clicking on "Chapter 1" will get you back to the main chapter graph.

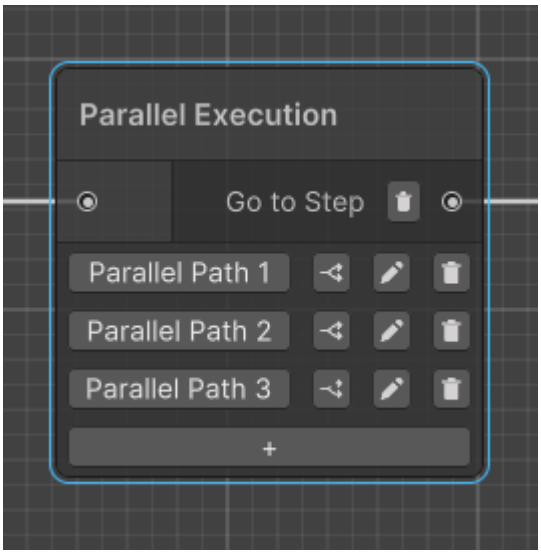
You can also create a group by selecting a sequence of nodes, right clicking and selecting `Make group`. Since the step group node only has one input and one output, this works best when selecting linked nodes only. Edge cases are resolved as follows:

- If there are two or more inbound connections in the group, all will lead to the group's input. The first valid node will be chosen as starting step for the group, while the others will have their connection severed.
- All outgoing connections will be deleted, meaning that the process will continue from the output of the group node after the group has processed. This means that if the selected nodes lead to multiple other nodes, now they will all go through the group's output.
- The step group output will be connected to the previous target of the first valid grouped node. Other external targets in grouped nodes are ignored, which means that when the group ends it will always go to the same following node.

If you encounter one of these edge case, make sure to review your process logic after grouping, as it may have changed.

Parallel Execution

The parallel execution node lets you execute two or more step sequences at the same time. Execution will continue to the next node once all parallel sequences have completed.



Clicking on a Parallel Path button will open a new graph where the path can be edited. This is very similar to a step group, with the difference that there can be multiple parallel paths and they are all executed at the same time. Like with step groups, it is possible to return to the main process by clicking the root chapter on the top left of the process editor.

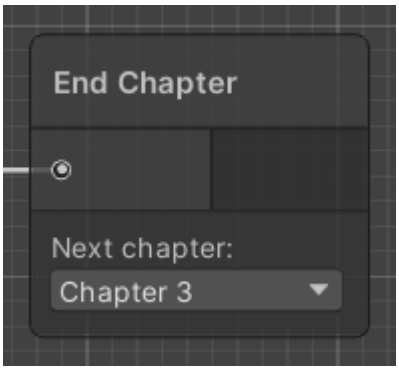
The buttons next to a parallel path let you make the path optional, rename it or delete it. Non-optional paths are displayed with the continuous arrow icon, while the interrupted arrow icon denotes a path that is optional.

When all non-optional paths have finished execution, the node ends execution as well. All optional paths are immediately interrupted. It can be useful to create optional paths to display looping animations, recurring hints and so on - they can even be endless loops without an exit point, as they will be interrupted anyway when the step ends.

The "+" button at the bottom lets you add more parallel paths. There is no theoretical limit to the number of paths in a parallel execution node, but a too high number might impact performance.

End Chapter

You can use this node as the last node on a sequence. It will immediately end the current chapter and start a new specified chapter, which can be selected from a drop-down list. This is useful to move through the chapters in a non-linear fashion. Note that you are not required to use this node for linear processes, as a chapter will automatically end when an empty transition is reached. In that case, the process will simply proceed to the following chapter.



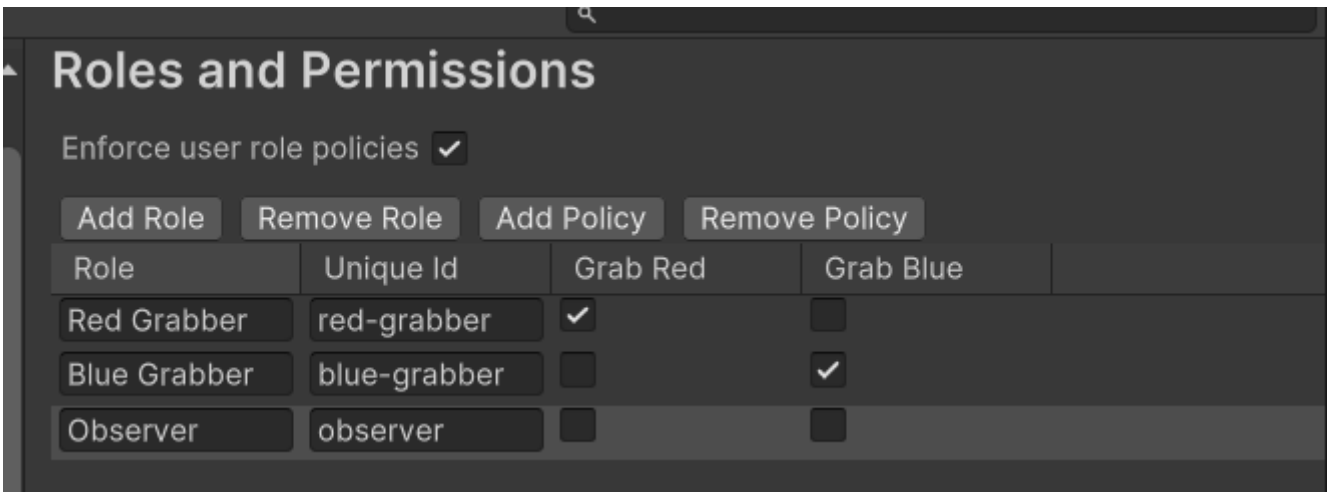
Permission Management

The VIROO Integration supports assigning roles to users to grant or limit permissions to the specified users. Each user can be in any number of groups. A role is a collection of policies, for example "Can grab objects", which gives a user with the role including the policy the ability to grab objects.

Currently the system allows to grant permissions to grab/touch objects (also supporting specific object groups), but can easily extended to add more policies and more control.

Configuration

The configuration page for roles and permissions can be found in `Project Settings > VR Builder > Roles and Permissions`. There, it is possible to define user roles and which policies are enforced for each of them.



The `Enforce user role policies` checkbox must be checked for roles and permissions to take effect. If it is checked, it is possible to add and remove roles and policies using the appropriate button.

A user role includes a display name and a unique id. The unique id is by default a GUID, but can be changed to something more human-readable as long as it stays unique.

Each user role also displays a checkbox for each policy that is enforced in the application. Ticking the checkbox grants the user role the permission handled by the policy.

Currently, the system does not support explicitly denying a permission. This means that, in case of overlapping policies, for example a generic "Grab Interactables" and a more specific "Grab objects in a specific group", as long as one of the two policies allow permission, the role will have permission to perform the action.

Assigning Roles

Two components are provided to assign roles to users in the scene.

SetDefaultUserRoles: This component is already present in the scene and can be found in the `Root > VR Builder > PERMISSION_HANDLER` game object. In the inspector, you can provide a list of user role unique ids. Every user connecting to the scene will be assigned these roles. For example, it can make sense to assign a "select role" role which allows them to interact with UI or objects that let them pick a role.

SetUserRolesOnInteract: You can add this component to an interactable object to assign the specified roles to users interacting with the object. In the inspector, can specify which interaction the user needs to perform (hover, select or activate) and which roles will be assigned, via a list of unique ids. Note that this will overwrite any roles the user already has, the idea being they are assuming a new role in the application. This component invokes an event when groups are assigned. This can be used e.g. to disable the object to represent that the role is already taken.

Policies

The VIROO Integration provides four policies to handle user permissions.

Grab Objects: The user is allowed to grab any interactable object.

Touch Objects: Same as above, but for touch permissions.

Grab Object Group: The user is allowed to grab Process Scene Objects assigned to the specified group.

Touch Object Group: Same as above, but for touch permissions.

It is possible to add new policies by overriding the `Policy` or `ConfigurablePolicy` abstract classes (or the base interface, in case more customization is needed). The `Policy` class is for basic policies, while the `ConfigurablePolicy` class can be tested against a parameter (for example to allow access to a specific object group as described above).

While overriding a simple `Policy` is fairly straightforward, there are a few caveat on implementing a `ConfigurablePolicy`.

- The parameter type must be serializable by Unity

- The policy must implement a `IEnumerable<T> GetPossibleValues()` method returning all values the parameter can have. Therefore, the policy parameter can only have a finite number of value and cannot be, for example, any float or custom string. This lets you select the configured policy from a submenu by clicking on the `Add Policy` button.

You as a developer are responsible for enforcing the policy by adding checks where relevant as explained below. For example, if you add a `EnterRestrictedAreaPolicy` policy, it is then up to you to code restricted areas that check for user permissions before allowing entry.

Checking for permissions

When implementing a new policy, you need to check for permissions when it's relevant to the policy. You can check whether a user has permission according to a specified policy by calling the Permission Handler. The Permission Handler is a centralized component which stores which user belongs to which groups across the network and can be accessed from the Runtime Configuration. A simple permission call looks like this:

```
IPermissionHandler permissionHandler =
RuntimeConfigurator.Configuration.GetPermissionHandler();
bool hasPermission = permissionHandler.UserHasPermission<PolicyType>(string
userId);
```

In the VIROO Integration, the user Id is the `IPlayer.ClientId` which can be accessed from the `IPlayer` component on the user's rig or from the `ISessionClientsProvider` injected object. In case of a configurable policy, for example the checks for process scene objects, you also need to specify a parameter in your request, for example:

```
IPermissionHandler permissionHandler =
RuntimeConfigurator.Configuration.GetPermissionHandler();
bool hasPermission =
permissionHandler.UserHasPermission<GrabProcessSceneObjectsPolicy, string>
(player.ClientId, objectGroupGuid.ToString());
```

Behaviors and Conditions

The VIROO Integration includes new behaviors and conditions designed to leverage VIROO's specific interactions and features. These include conditions to check interactions with controls like levers, buttons and switches, matching the interactable prefabs found in VIROO Studio. It also includes a behavior that executes VIROO Actions. The new behaviors and conditions can be found in the VIROO section of the respective menus.

Execute VIROO Actions Behavior

This behavior execute the first VIROO action on every referenced object. It can be handy to use VIROO actions for functionalities not present in the VR Builder behavior library. Additionally, since actions are natively networked, they can be particularly useful in multi-user apps. It is important to note that, since not all VIROO

actions notify when the action has been completed, this behavior triggers the action and then deactivates, potentially leading to ending the step immediately.

Parameters

TargetObjects: References to the game objects containing the actions. Note that in case multiple action components are present on the same game object, only the first one will be executed.

Grip Button Condition

This condtion completes when the user grips the specified grip button(s).

Parameters

Controls: References to the grip buttons checked by this condition.

Trigger on release: If this is checked, the condition will complete only when the user releases the button. Otherwise, it will complete as soon as the button is gripped.

Trigger all controls: If checked, the condition completes when all referenced controls have been interacted with. Otherwise, the user needs to interact with only one of the listed controls in order to complete the condition.

Pull Lever Condition

This condition completes when the user pulls the specified lever(s), setting it either to its true or false position.

Parameters

Controls: References to the levers checked by this condition.

Target position: The specified target position of the lever, either true or false.

Require release: If this is checked, the condition will complete only when the user releases the lever. Otherwise, it will complete as soon as the lever reaches the desired state.

Set all controls: If checked, the condition completes when all referenced controls have been interacted with. Otherwise, the user needs to interact with only one of the listed controls in order to complete the condition.

Push Button Condition

This condtion completes when the user pushes the specified button(s).

Parameters

Controls: References to the buttons checked by this condition.

Trigger on release: If this is checked, the condition will complete only when the user stops pushing the button. Otherwise, it will complete as soon as the button is pushed.

Trigger all controls: If checked, the condition completes when all referenced controls have been interacted with. Otherwise, the user needs to interact with only one of the listed controls in order to complete the

condition.

Set Slider Condition

This condition completes when the specified slider(s) are set within a specified range.

Parameters

Controls: References to the sliders checked by this condition.

Min position: The lower end of the range that will cause the condition to complete, on a scale from 0 to 1.

Max position: The higher end of the range that will cause the condition to complete, on a scale from 0 to 1.

Require release: If this is checked, the condition will complete only when the user releases the slider. Otherwise, it will complete as soon as the slider reaches the desired position.

Set all controls: If checked, the condition completes when all referenced controls have been interacted with. Otherwise, the user needs to interact with only one of the listed controls in order to complete the condition.

Turn Knob Condition

This condition completes when the specified knob(s) are set within a specified range.

Parameters

Controls: References to the knobs checked by this condition.

Min position: The lower end of the range that will cause the condition to complete, on a scale from 0 to 1.

Max position: The higher end of the range that will cause the condition to complete, on a scale from 0 to 1.

Require release: If this is checked, the condition will complete only when the user releases the knob. Otherwise, it will complete as soon as the knob reaches the desired position.

Set all controls: If checked, the condition completes when all referenced controls have been interacted with. Otherwise, the user needs to interact with only one of the listed controls in order to complete the condition.

Limitations

- Some VR Builder Pro features, such as data properties, are currently not networked but can be used as long as all the value reading and writing is done by the process, as this ensure it is all done on the same machine. This means you should not write custom code that assigns a value outside VR Builder since that won't be replicated if it happens on a different client.
- The Teleport Condition only works in single user at the moment.
- Currently, the VR Builder process will run on the first client that connects to the session. Disconnecting that client will break the process.

Multi-User

Applications created with this integration can work in multi-user through the VIROO Player the same way a normal VIROO application does.

The VR Builder process runs on a single client. Interactions from other clients are replicated and can advance the process seamlessly. Behaviors triggered by the process (e.g. text-to-speech, animations) are replicated on all clients.

The VR Builder process starts automatically as soon as the first user joins the scene.

Existing behaviors and conditions are designed to work seamlessly in multi-user, but developers should ensure their custom code supports running on different clients as well.

Update Strategy

When updating the VR Builder packages from the Asset Store (e.g. VR Builder Pro), it is recommended to remove the package from the project before re-importing it. This prevents conflicts with deleted or renamed files between versions.

Best Practices

The VIROO Integration is designed to be a seamless experience for users which are already familiar with VR Builder. Therefore, most of the best practices applicable to the design of VR Builder processes in the default environment also apply while working in VIROO Studio. Additional care should be given to the specific requirements of the VIROO platform. The following section provides an introduction to working with VR Builder in the context of the VIROO integration. It covers how to set up a project correctly, best practices for testing during development, and how to customize VR Builder to your needs.

Project Setup

It is good practice to ensure that the project has been set up correctly. Ensure you are not using a scriptable render pipeline as VIROO Studio 2.6 only supports Unity's built-in render pipeline. The following packages should have been imported in a VIROO Studio 2.6 template project:

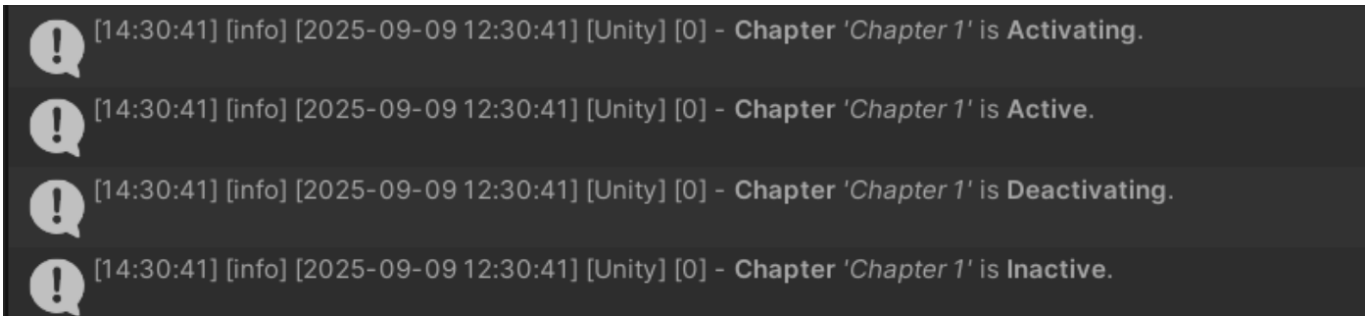
- VR Builder Core 5.3.1 or later
- Roles and Permissions for VR Builder 1.1.0 or later
- VIROO Integration for VR Builder 1.1.1. or later

If all packages are present and no errors are displayed, then the project is ready for VR app creation. You can test that everything works as intended by creating an empty VR Builder scene and ensuring the process runs as expected.

- Open the automated setup wizard from Tools > VR Builder > Scene Setup Wizard...
- Give a name to your test scene and process, and ensure *VIROO* is selected in the scene type dropdown. Leave both *Create a new process* and *Setup the scene for VR Builder* ticked.
- Confirm to create the scene. An empty scene should be created.
- Press Play.

When pressing Play, the VIROO avatar should appear in the scene. In the console, log messages similar to the following should be displayed.

This means that the (empty) VR Builder process is running correctly.



You can now start creating your VR application!

Working with VR Builder

VR Builder excels at creating applications where the user needs to perform a sequence of actions in a specific order. The sequence does not necessarily need to be linear, it can be branching and it is indeed possible to create more freeform scenarios, but the logic is based on expecting specific user actions and performing corresponding reactions.

Additionally, VR Builder automatically locks process objects that are not relevant to the current step, and unlocks the ones the user should be interacting with to keep the focus on the process. This means it is not recommended to use VR Builder to model environmental interactions that might be available all the time. For example, if your scene features a light that can be turned on and off at any time, possibly as a cosmetic environment prop or a tool needed by the user at unspecified times, then we don't recommend implementing it through VR Builder. While possible, it is going to be needlessly cumbersome. On the contrary, if the user is supposed to turn on the light at some point during the process, and not touch it otherwise, then it is a good idea to create a "Turn on light" step in the VR Builder process which will handle the interaction and following environment change.

This is twice as relevant while working with VIROO Studio, as the latter offers the tools to quickly create interactions like this in the scene without needing to code.

In short, we recommend using VR Builder for the linear process, i.e. the sequence of actions the user needs to perform, and using standard VIROO actions and interactions for the parts of the environment that need to be interactable at all times without being part of the VR Builder process.

It is worth noting that VR Builder will only lock objects that have a Process Scene Object component, that is objects that are referenced in the process. Interactable objects without that component are not recognized by VR Builder and will not be locked while the application is running, so the user will be able to interact with them at all times.

Testing VR Builder processes

Testing a VR Builder process can be daunting, requiring the user to go through all steps to check out the latest interaction. The VR Builder Pro package provides menu prefabs that can be used to skip steps and even skip chapters to reach the relevant chapter, so it can make things easier. This improves if the process is split into logically sensible chapters and step groups.

Users who do not own VR Builder Pro (which is not required to run the VIROO Integration) can achieve similar effects by temporarily modifying the process. Attaching an End Chapter node as the very first step of the first

chapter can instantly bring the tester to the relevant chapter. Steps that do not need to be tested can also be temporarily skipped in the graph.

Additionally, VR Builder provides extensive logging options to assess which step is running and give actionable information on how to solve possible problems. You can adjust the logging options to be as granular as you like in Project Settings > VR Builder > Logging window.

Logging Entities

The first five options log the "building blocks" of a process. To that end, we need to understand the structure of the process:

- A Process contains one or more Chapters.
- A Chapter contains a number of Steps.
- A Step contains Behaviors and Transitions, as is visible in the Step Inspector.
- A Transition can have one or more Conditions.

All these elements follow the same logic and go through a lifecycle that starts from Inactive and goes through the *Activating*, *Active*, and *Deactivating* states.

Log messages will log changes in these states. Broadly speaking, these can be interpreted as follows.

- *Activating* entities are waiting for something to execute. An activating chapter is running through the steps it contains, an activating step is executing its behaviors, and so on.
- *Active* entities are waiting for user input. Usually, this relates to active conditions waiting to be fulfilled, e.g. a grab condition waiting for the user to grab an object. Entities that are not conditions usually switch away from the active states immediately.
- *Deactivating* entities are again waiting for something to execute, before turning inactive. This is mostly used by steps executing some behaviors before the step ends. Most entities deactivate immediately.

You can use this knowledge to better debug your process. For example, if the process seems stuck but the log says that a specific condition is active, it means that for some reason that condition is not being fulfilled.

Logging Data Property changes

This option logs when the value stored in a Data Property changes. Data Properties are VR Builder's version of variables and this helps you track any potential bugs related to their expected values.

Logging Lock/Unlock requests

This logs when VR Builder locks or unlocks a process scene property. VR Builder only makes interactable (unlocks) the process objects that are needed in the current step, and only for the interaction expected by the step. It is possible for multiple parallel steps to unlock the same property, and for developers to manually unlock a property in the desired steps. This tool ensures you can correctly visualize the lock state of an object - useful when you are not able to interact with an object as expected.

Extending VR Builder

A key strength of VR Builder is its ability to be customized with your own behaviors and conditions. We strongly recommend exploring this option when facing technical challenges. Often, a simple custom behavior or condition can eliminate complex workarounds and ultimately save you considerable time.

Furthermore, you can design reusable custom behaviors and conditions, allowing you to build a library of tailored building blocks that streamline your workflow over time.

You can find tutorials on how to build custom behaviors and conditions on our website, at the following links.
How to create custom behaviors: https://www.mindport.co/vr-builder-tutorials/creating-custom-behaviors
How to create custom conditions: https://www.mindport.co/vr-builder-tutorials/creating-custom-conditions

Behaviors and conditions are not the only extendable elements in VR Builder, and we recommend checking the official documentation for more information on how it is possible to customize the development environment.