Tutorial

- Preparing Scenes and Views
- Screen Transitions with View Tree
- Structuring Screen Transitions with Parent-Child View States
- Controlling Transitions with Conditions and Action Chains

Preparing Scenes and Views

This page covers the preparation of the foundation used throughout the subsequent pages. Starting from the Empty solution, you will set up a learning environment, configure the Render Targets, and create the basic scenes — Background, Header, MainContent, and DetailContent — along with the corresponding Views for each.

In the subsequent pages, you will use these elements to learn about screen switching with sibling Views, screen structuring with parent-child Views, and conditional transitions using the Action Chain Editor and Condition Editor.

Preparing the solution

In this task, you prepare a learning environment free from the influence of existing samples. If unnecessary scenes or State Machines remain from the start, they can be difficult to distinguish from the structures you are about to create, so it is important to clean them up first.

1. Create a new solution from the sample solution Empty.

2. In the Solution Explorer, delete the Scene2D and Scene3D scenes from Scenes.

3. Select Render Targets under Scenes and configure the following properties in the Properties panel:

Property Category	Item	Value
Render Target	Auto Swap	True
	Auto Clear	True

4. Next, delete the GlobalStateMachine from the State Machine in the Solution Explorer.

5. Save the solution with a name of your choice.

This configuration enables the system to update the display all at once after the entire Render Target finishes drawing, rather than reflecting each camera's rendering to the display individually. However, to prevent afterimages from the previous screen, the critical factor is ensuring that the Background scene (created next) correctly handles color buffer clearing.

Background Scene

The purpose of this task is not simply to place a background. The Background scene plays the role of finalizing the screen's color content at the beginning of every frame. If this scene is not configured correctly, the previous screen might appear to remain visible, even though the View switch itself succeeds.

1. Create a new 2D Scene and name it **Background**.

2. Select the camera in the Scene Tree and configure the following properties in the Properties panel:

Property Category	Item	Value
Camera	Sequence Number	0
	Clear Color Buffer	True
	Enable Swapping	False

3. Place a 2D Text Node from the Toolbox onto the Scene Tree of the Background scene.

Property Category	Item	Value
Transform	Position X	0
	Position Y	0
2D Text Node	Text	Background

The Background scene requires Clear Color Buffer to be set to True. This clears the previous color content before drawing a new frame, so even when subsequent content scenes overlay their rendering with Clear Color Buffer set to False, the screen is less likely to appear doubled. Note that because the Background scene serves as both the shared background and the clearing handler, it is safer not to include screen-specific decorations in this scene.

Header Scene

Here, you create a shared display element that multiple screens reuse. The Header is an element displayed in both MainView and DetailView, so you will later extract it as an independent View for reuse.

1. Create a new 2D Scene and name it **Header**.

2. Select the camera in the Scene Tree and configure the following properties in the Properties panel:

Property Category	Item	Value
Camera	Sequence Number	2
	Clear Color Buffer	False
	Enable Swapping	False

3. Place a 2D Text Node from the Toolbox onto the Scene Tree of the Header scene.

Property Category	Item	Value
Transform	Position X	300
	Position Y	10
2D Text Node	Text	Common Header

The Header is a layer that simply overlays on top of the background and content. If you clear the color buffer here, you might erase previously drawn elements. By assigning a Sequence Number that places the Header in front of the background and the content, the Header achieves a natural stacking order as a heading.

MainContent Scene

In this task, you create the main screen content used for the first screen switch. The buttons you add here serve as the entry point for building sibling View transitions in the next page.

1. Create a new 2D Scene and name it **MainContent**.

2. Select the camera in the Scene Tree and configure the following properties in the Properties panel:

Property Category	Item	Value
Camera	Sequence Number	1
	Clear Color Buffer	False
	Enable Swapping	False

3. Place a 2D Text Node from the Toolbox onto the Scene Tree of the MainContent scene.

Property Category	Item	Value
Transform	Position X	50
	Position Y	50
2D Text Node	Text	Main Screen

4. Next, add two TextButton controls and configure each as follows:

Property Category	Item	Value
TextButton 1		
General	Name	OpenDetailButton
Transform	Position X	100
	Position Y	100
Control	Text	Open Detail
TextButton 2		
General	Name	OpenNavigationButton
Transform	Position X	400
	Position Y	100
Control	Text	Open Navigation

MainContent is a standard content scene placed on top of the background, so Clear Color Buffer remains set to False. The Background scene handles the clearing, and this scene only overlays the screen content on top of that background.

DetailContent Scene

Here, you create the content for the DetailView. The Back button is used for return transitions between sibling Views, and the CheckBox is used as a control input for conditional automatic transitions in a later page.

1. Create a new 2D Scene and name it **DetailContent**.

2. Select the camera in the Scene Tree and configure the following properties in the Properties panel:

Property Category	Item	Value
Camera	Sequence Number	1
	Clear Color Buffer	False
	Enable Swapping	False

3. Place a 2D Text Node from the Toolbox onto the Scene Tree of the DetailContent scene.

Property Category	Item	Value
Transform	Position X	50
	Position Y	50
2D Text Node	Text	Detail Screen

4. Next, place a TextButton control and configure the following properties:

Property Category	Item	Value
General	Name	BackButton
Transform	Position X	100
	Position Y	100
Control	Text	Back

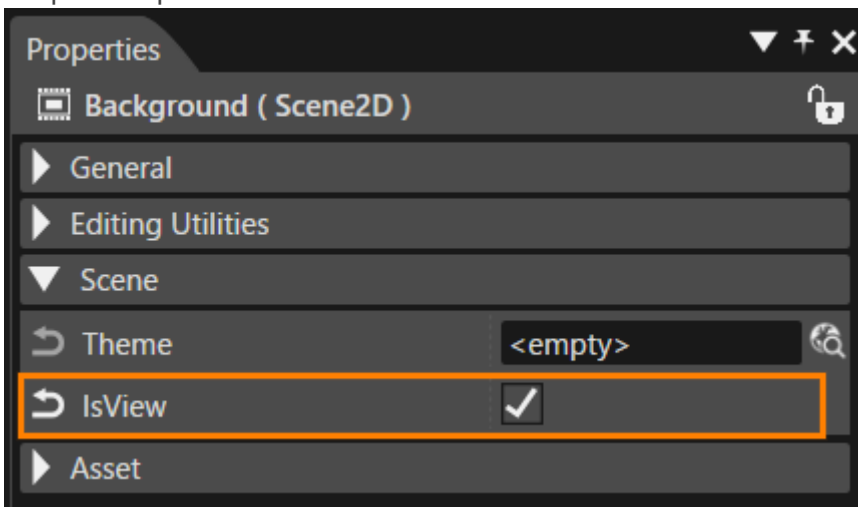
5. Finally, add a CheckBox control and configure the following properties:

Property Category	Item	Value
General	Name	KeepDetailOpenCheckBox
Transform	Position X	100
	Position Y	200
Control	Text	Keep Detail Open

Configuring Scenes as Views

At this stage, you make the foundational components used in the subsequent pages referenceable as Views. When you set a Scene's IsView property to true, that Scene becomes referenceable as a View.

1. Select each of the four scenes created in the previous steps (Background / Header / MainContent / DetailContent) and select the IsView checkbox in the Scene category of the Properties panel.



Because the Scene name is used as the View Identifier as is, specify the same names (Background / Header / MainContent / DetailContent) when referencing them from the View Definition Editor in the subsequent pages.

In the next page, you will combine these Views to compose MainView and DetailView, and learn how to switch between them as sibling relationships on the View Tree.

Screen Transitions with View Tree

In this page, you use the basic Views prepared in the previous page to compose two screen-level Views, MainView and DetailView, and arrange them as siblings in the View Tree.

The central themes of this page are how to assemble a single screen in the View Definition Editor and why a sibling relationship in the View Tree is suited for mutually exclusive screen switching. You also configure Jump To View on a button and verify in the Player that MainView and DetailView switch without overlapping. This page is the core part for learning the basics of the View Editor — defining screen-level Views and using sibling View transitions.

Creating MainView

Here, you bundle the background, the common header, and the main content into a View that represents the entire main screen. Proceed while understanding that the View Definition Editor is "the place to assemble a screen-level View."

1. In Solution Explorer, select the Solution folder and select **Add > New Item** from the context menu.

2. In the **New Item** dialog, select View and name it MainView.

3. Double-click the created MainView to open the View Definition Editor.

4. Add the following elements to the left pane of the View Definition Editor:
   - Background
   - Header
   - MainContent

MainView is a View that bundles the background, the common header, and the contents of the main screen into a single screen state. It is also important that you reuse Header as an existing View — this teaches the idea of extracting shared components as separate Views and reusing them.

Creating DetailView

Next, you define a View that represents the entire detail screen. As with MainView, you use the background and header as shared components, and only the content portion differs. This lets multiple screens share the same visual framework while only switching their content.

1. In Solution Explorer, select the Solution folder and select **Add > New Item** from the context menu.

2. In the **New Item** dialog, select View and name it DetailView.

3. Double-click the created DetailView to open the View Definition Editor.

4. Add the following elements to the left pane of the View Definition Editor:
   - Background
   - Header
   - DetailContent

DetailView is also an independent View that represents the entire screen. Because it shares the same background and header as MainView and only replaces the content, you can easily perform screen-level switching in the View Tree later.

Configuring MainViewTree

Here, you place MainView and DetailView in the View Tree. The most important point is to make the two a sibling relationship. A sibling relationship is suited for switching between independent screen states.

1. In Solution Explorer, select the Solution folder and select **Add > New Item** from the context menu.

2. In the **New Item** dialog, select ViewTree and name it MainViewTree.

3. Double-click the created MainViewTree to open the View Tree panel.

4. Drag and drop MainView onto the View Tree panel to add it as a top-level View State.

5. In the same way, drag and drop DetailView into the empty area below MainView to add it as a top-level View State.

Verifying the Sibling Relationship in the View Tree

- MainView and DetailView are at the same hierarchical level (DetailView is not a child of MainView).
- The two are targets to switch between exclusively.

> If you drop DetailView onto the body of MainView here, you create a parent-child relationship, and the resulting structure may unintentionally behave close to simultaneous display. Because you want to switch between the entire screens this time, always drop into the empty area below to place them as siblings.

Configuring Transitions Between Views

From here, you make it possible to move between the sibling Views defined in the View Tree using Jump To View. The key point is that you specify the completed screen-level View as the transition target, not a content View.

1. Double-click the MainContent scene to open the Scene Editor.
2. In the Scene Tree, select OpenDetailButton.
3. Drag and drop a Jump To View behavior onto Action in the Extra Scene Tree.
4. Select the added Jump To View behavior and configure the following properties in the Properties panel:

Property Category	Item	Value
General	Name	JumpToDetailView
Asset	View	DetailView

You specify DetailView (a Composite View that includes the background and header) — not DetailContent (a Scene with IsView=true on its own) — as the transition target because you want to switch at the screen level, including the background and header. The flow of treating a Composite View created in the View Definition Editor as a state in the View Tree connects here.

In the same way, configure the return transition from DetailView to MainView. This completes the round-trip switching between sibling Views.

1. Double-click the DetailContent scene to open the Scene Editor.
2. In the Scene Tree, select BackButton.
3. Drag and drop a Jump To View behavior onto Action in the Extra Scene Tree.
4. Select the added Jump To View behavior and configure the following properties in the Properties panel:

Property Category	Item	Value
General	Name	JumpToMainView
Asset	View	MainView

Verifying in the Player

Here, you verify not just the switching between MainView and DetailView, but also whether the Background's clear configuration is effective. Even when the transition itself succeeds, if the previous screen remains visible, the cause is more often on the Background side than in the View Tree.

1. Save the solution and launch the Player.
   - Verify that MainView is displayed immediately after launch.

2. Click the Open Detail button.
   - Verify that the screen switches to DetailView.

3. Click the Back button.
   - Verify that the screen returns to MainView.

In this page, you composed MainView and DetailView as screen-level Views in the View Definition Editor and placed them as siblings in the View Tree. You also implemented basic screen switching between sibling Views using Jump To View. In the next page, you learn a different use case — a parent-child relationship that switches only a local state while preserving the shared display.

Structuring Screen Transitions with Parent-Child View States

In this page, you define NavigationView as a parent Composite View, and inside it, you build a structure that switches between NavigationMain (a Scene with IsView=true) and RouteDetailsView (a Composite View) as child View States. What you learn here is that, unlike the sibling relationship in the previous page, a parent-child relationship is suited for "switching only a local state while maintaining the shared display." By switching between the normal display and the detail display while maintaining the map and shared buttons inside NavigationView, you understand the practical use of the parent-child structure in the View Tree. You also revisit the composition that combines a Composite View with a Scene that has IsView=true.

Creating Shared Scenes for Navigation

First, you prepare the elements that you want to display continuously while NavigationView is active. Here, you handle the map display and the Home button as shared parts, and incorporate them into the parent View definition later.

NavigationMap Scene

1. Create a new 2D scene and name it NavigationMap.

2. Select the camera in the Scene Tree and configure the following properties in the Properties panel:

Property Category	Item	Value
Camera	Sequence Number	1
	Clear Color Buffer	False
	Enable Swapping	False

3. Place a 2D Text Node from the Toolbox onto the Scene Tree of the NavigationMap scene.

Property Category	Item	Value
Transform	Position X	250
	Position Y	200
2D Text Node	Text	Navigation Map

NavigationViewButtons Scene

1. Create a new 2D scene and name it NavigationViewButtons.

2. Select the camera in the Scene Tree and configure the following properties in the Properties panel:

Property Category	Item	Value
Camera	Sequence Number	2
	Clear Color Buffer	False
	Enable Swapping	False

3. Place a TextButton control from the Toolbox onto the Scene Tree of the NavigationViewButtons scene.

Property Category	Item	Value
General	Name	JumpToHomeViewButton
Transform	Position X	350
	Position Y	300
Control	Text	Home

NavigationMap and NavigationViewButtons are elements that you want to remain visible while in NavigationView. Therefore, you incorporate them into the parent View definition later.

Creating Transition Scenes for Navigation

Next, you prepare the targets to switch between inside NavigationView. Here, you separate the normal display and the route details display so that you can treat them as child states of the parent View.

NavigationMain Scene

1. Create a new 2D scene and name it NavigationMain.

2. Select the camera in the Scene Tree and configure the following properties in the Properties panel:

Property Category	Item	Value
Camera	Sequence Number	2
	Clear Color Buffer	False
	Enable Swapping	False

3. Place a 2D Text Node from the Toolbox onto the Scene Tree of the NavigationMain scene.

Property Category	Item	Value
Transform	Position X	50
	Position Y	200
2D Text Node	Text	Navigation Main

4. Next, place a TextButton control and configure the following properties:

Property Category	Item	Value
General	Name	ShowDetailsButton
Transform	Position X	50
	Position Y	300
Control	Text	Show Details

RouteDetailsContent Scene

1. Create a new 2D scene and name it RouteDetailsContent.

2. Select the camera in the Scene Tree and configure the following properties in the Properties panel:

Property Category	Item	Value
Camera	Sequence Number	2
	Clear Color Buffer	False
	Enable Swapping	False

3. Place a 2D Text Node from the Toolbox onto the Scene Tree of the RouteDetailsContent scene.

Property Category	Item	Value
Transform	Position X	450
	Position Y	200
2D Text Node	Text	Route Details

NavigationBack Scene

1. Create a new 2D scene and name it NavigationBack.

2. Select the camera in the Scene Tree and configure the following properties in the Properties panel:

Property Category	Item	Value
Camera	Sequence Number	2
	Clear Color Buffer	False
	Enable Swapping	False

3. Place a TextButton control from the Toolbox onto the Scene Tree of the NavigationBack scene and configure the following properties:

Property Category	Item	Value
General	Name	BackToNavigationMainButton

Transform	Position X	50
	Position Y	300
Control	Text	Back

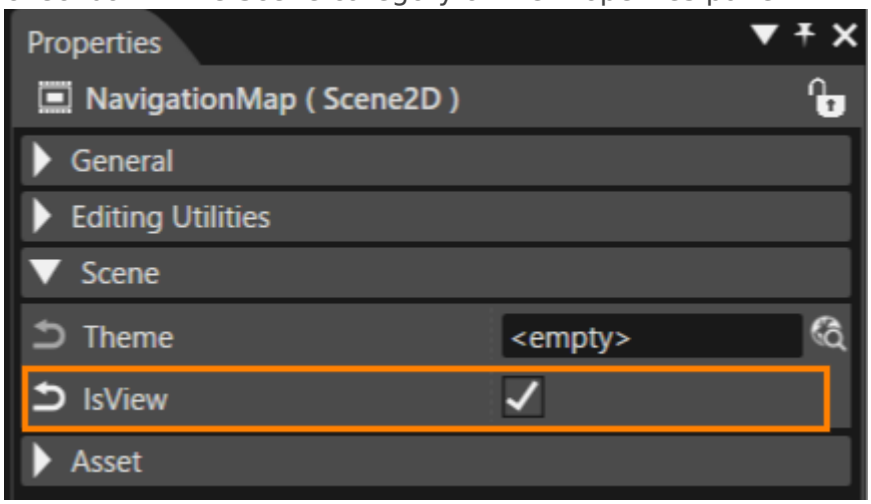
NavigationMain is for the normal display, and RouteDetailsContent and NavigationBack are components for the detail display. These are not shared displays — you treat them as targets to switch between inside NavigationView.

Creating Navigation-Related Views

Here, you use both the case of creating a simple View from an existing Scene and the case of creating a new View that combines multiple components. This distinction is also important when learning the parent-child structure.

Creating Views: Using Existing Scenes

1. Select each of the five scenes created in the previous steps and select the IsView checkbox in the Scene category of the Properties panel.



- NavigationMap
- NavigationViewButtons
- NavigationMain
- RouteDetailsContent
- NavigationBack

Creating the New RouteDetailsView

1. Create a new View and name it RouteDetailsView.

2. Double-click the created RouteDetailsView to open the View Definition Editor.

3. Add the following elements to the left pane of the View Definition Editor:
   - RouteDetailsContent
   - NavigationBack

RouteDetailsView is a screen state that bundles the detail display itself and the return operation. By extracting a local grouping like this as a Composite View, you make it easier to treat it as a child View State later.

Configuring NavigationView

Here, you bundle into the parent View only the elements that you want to remain displayed while in Navigation. It is important to include only the shared display in the parent View, and not the parts that you want to switch.

1. In Solution Explorer, select the Solution folder and select **Add > New Item** from the context menu.

2. In the **New Item** dialog, select View and name it NavigationView.

3. Double-click the created NavigationView to open the View Definition Editor.

4. Add the following elements to the left pane of the View Definition Editor:
   - Background
   - Header
   - NavigationMap
   - NavigationViewButtons

You do not include NavigationMain or RouteDetailsView in the NavigationView definition. You add these later as child View States in the View Tree, and treat them as targets to switch between under the parent. This separation clarifies the roles of the shared display and the local switching.

Configuring MainViewTree

Here, you add NavigationView as a top-level sibling View and attach child View States inside it. This lets you experience a two-layer structure — sibling relationships at the top level and a parent-child relationship inside Navigation.

1. Double-click MainViewTree to open the View Tree panel.

2. Drop NavigationView into the empty area below MainView and DetailView to add it as a top-level View State.

3. Drop NavigationMain onto the body of the parent NavigationView to add it as a child View State.

4. Drop RouteDetailsView onto the body of the parent NavigationView to add it as a child View State.

By dropping onto the body of the parent, the child states are treated as internal states of NavigationView while it is active. This is the essence of a parent-child relationship, as opposed to a sibling relationship.

Configuring Navigation-Related Transitions

Here, you configure both the transitions to top-level sibling View States and the switching of child View States under the parent. This shows that the same Jump To View takes on different meanings depending on the relationship of the View you jump to.

MainContent Scene

1. Double-click the MainContent scene to open the Scene Editor.

2. In the Scene Tree, select OpenNavigationButton.

3. Drag and drop a Jump To View behavior onto Action in the Extra Scene Tree.

4. Select the added Jump To View behavior and configure the following properties in the Properties panel:

Property Category	Item	Value
General	Name	JumpToNavigation
Asset	View	NavigationView

NavigationMain Scene

1. Double-click the NavigationMain scene to open the Scene Editor.

2. In the Scene Tree, select ShowDetailsButton.

3. Drag and drop a Jump To View behavior onto Action in the Extra Scene Tree.

4. Select the added Jump To View behavior and configure the following properties in the Properties panel:

Property Category	Item	Value
General	Name	JumpToRouteDetail
Asset	View	RouteDetailsView

NavigationBack Scene

1. Double-click the NavigationBack scene to open the Scene Editor.
2. In the Scene Tree, select BackToNavigationMainButton.
3. Drag and drop a Jump To View behavior onto Action in the Extra Scene Tree.
4. Select the added Jump To View behavior and configure the following properties in the Properties panel:

Property Category	Item	Value
General	Name	JumpToNavigationMain
Asset	View	NavigationMain

NavigationViewButtons Scene

1. Double-click the NavigationViewButtons scene to open the Scene Editor.
2. In the Scene Tree, select JumpToHomeViewButton.
3. Drag and drop a Jump To View behavior onto Action in the Extra Scene Tree.
4. Select the added Jump To View behavior and configure the following properties in the Properties panel:

Property Category	Item	Value
General	Name	JumpToMainView
Asset	View	MainView

The important point here is that, inside NavigationView, NavigationMain and RouteDetailsView switch between each other as child View States, while the Home button exits Navigation entirely and returns to MainView, the top-level sibling View State.

Verifying in the Player

The purpose of this verification is to see how the meaning of the parent-child relationship is reflected in the screen display. Verify that the shared display portion is maintained while only the child state switches.

1. Save the solution and launch the Player.
   - Verify that MainView is displayed immediately after launch.

2. Click the Open Navigation button.
   - Verify that the screen switches to NavigationView.
   - At the same time, verify that Background and Header are displayed.
   - In addition, verify that Navigation Main, Navigation Map, the Show Detail button, and the Home button are displayed.

3. Click the Show Details button.
   - Verify that the screen switches to Route Details.
   - At the same time, verify that Background and Header are displayed.
   - In addition, verify that Navigation Map and the Home button are displayed.

4. Click the Back button.
   - Verify that the screen returns to Navigation Main.

5. Click Home.
   - Verify that the screen returns to MainView.

In this page, you built a structure that uses NavigationView as the parent View State and switches between child View States inside it. This let you confirm in practice that a sibling relationship is suited for exclusive switching of entire screens, while a parent-child relationship is suited for switching only a local state while retaining the shared display. In the next page, you add conditional automatic transitions to DetailView using the Action Chain Editor and the Condition Editor.

Controlling Transitions with Conditions and Action Chains

In this page, you implement a conditional automatic transition for DetailView, combining a timer with a checkbox value. The central theme is the division of roles: the Action Chain Editor is the place to build the processing flow, while the Condition Editor is the place to define the condition expression independently. You build behavior that returns to MainView after a certain time elapses after entering DetailView, while stopping the auto-transition only when the Keep Detail Open CheckBox is on. This lets you understand the coordination of triggers, conditions, and actions in practice. In particular, in this page, setting OperationMode of Check Processed Value to CompareExternalValueWithConditionValue is the key to correct behavior.

Configuring the CheckBox Control

First, you convert the state of the CheckBox control placed in the DetailContent scene into a form readable by the Condition Editor. If you skip this step, you cannot use the current checked state in a condition expression.

1. Double-click the DetailContent scene to open the Scene Editor.

2. In the Scene Tree, select KeepDetailOpenCheckBox.

3. Drag and drop a Forward Value behavior onto the Extra Scene Tree.

4. Select the added Forward Value behavior and configure the following properties in the Properties panel:

Property Category	Item	Value
Asset	EventDispatchStrategy	DedicatedBehavior

5. Add a Value behavior under DedicatedBehavior of the Forward Value behavior.

6. Drag and drop the added Value behavior onto the DetailContent scene in the Scene Tree to associate it.

With this configuration, the current value of the CheckBox control is held on the scene node, and the Condition Editor can reference it as an external value. The target that you specify as ValueBehavior in the next step is the Value you created here.

Configuring the ConditionBlock

Here, you define the condition logic that determines whether the automatic transition is allowed, separately from the Action Chain. Separating the condition makes it easier to understand the processing flow and the evaluation logic independently.

In this task, you build a condition that becomes true only when both the timer has fired and the Keep Detail Open CheckBox is off. This is the heart of the conditional transition.

1. In Solution Explorer, select the Solution folder and select **Add > New Item** from the context menu.

2. In the **New Item** dialog, select ConditionBlock and name it ReturnToMainOnTimeoutCondition.
   - The Condition Editor panel appears and displays the diagram of the newly created ReturnToMainOnTimeoutCondition > Condition.

3. On the diagram, use the context menu and select **Add Check > Condition > Control > Check Timer** to add it to the diagram.

4. In the same way, from the context menu, select **Add Check > Condition > Control > Check Processed Value** to add it to the diagram.

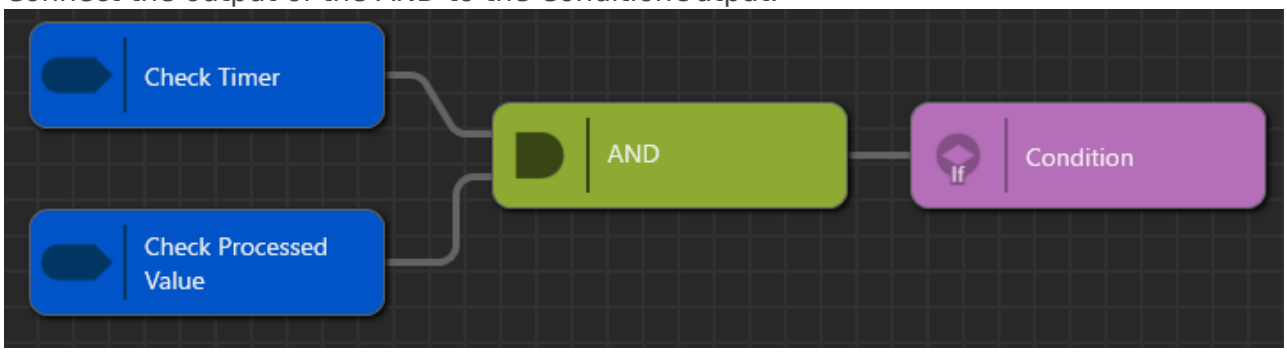
5. Select the added Check Processed Value and configure the following properties in the Properties panel:

Property Category	Item	Value
Asset	OperationMode	CompareExternalValueWithConditionValue
	ConditionOperator	Equal
	ConditionValue type	Bool
	ConditionValue	False
	ValueBehavior	DetailContent scene > Value

6. Next, from the context menu, select **Add AND** to add an AND node.

7. Connect Check Timer and Check Processed Value to the AND.

8. Connect the output of the AND to the ConditionOutput.



The meaning of the condition built here is: allow the automatic transition only when the timer has fired and the checkbox is off. The OperationMode setting is especially important. If you set it to an in-process event-dependent mode such as CompareProcessedValueWithConditionValue, no value event to compare against exists at the time of the Timer event, and the condition is not evaluated correctly. The correct choice here is CompareExternalValueWithConditionValue, which references the external value held on the scene node.

Configuring On Entry for DetailView

Next, configure the action that starts the timer at the moment of entering DetailView. At this point, no transition occurs yet — you only set up the reference time for the subsequent condition evaluation.

You can edit a View's OnEntry action in the OnEntry / OnExit Editing Mode of the Action Chain Editor. This tutorial uses this method.

1. Double-click MainViewTree to open the View Tree panel.

2. Double-click DetailView to open the Action Chain Editor.
3. In the left pane of the Action Chain Editor, double-click State > OnEntry.
   - The OnEntry pseudo-block appears in the right pane.

4. On the right side of the pseudo-block, select **Add Action > Action > Control > Start Timer** from the context menu to add a Start Timer action.

5. Select the added Start Timer and configure the following properties in the Properties panel:

Property Category	Item	Value

Asset	Delay	3000

This configuration means: use the moment three seconds after entering DetailView as the timing for the automatic transition decision. Note that starting the timer and the transition itself are separate configurations. The content added to OnEntry in the Action Chain Editor is automatically synchronized with the OnEntry property of DetailView.

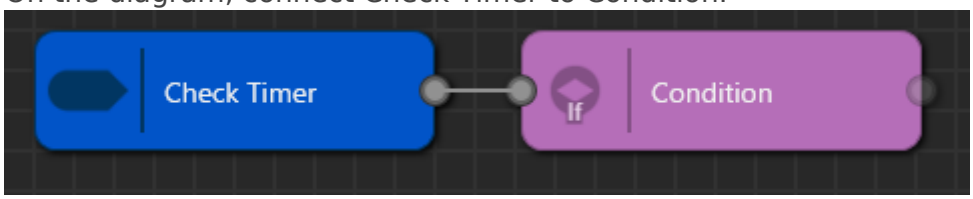
Configuring the Action Chain

Finally, build a processing flow that receives the timer event, evaluates the ConditionOutput, and returns to MainView only when the result is true. This is the first point at which the Action Chain Editor and the Condition Editor work together.

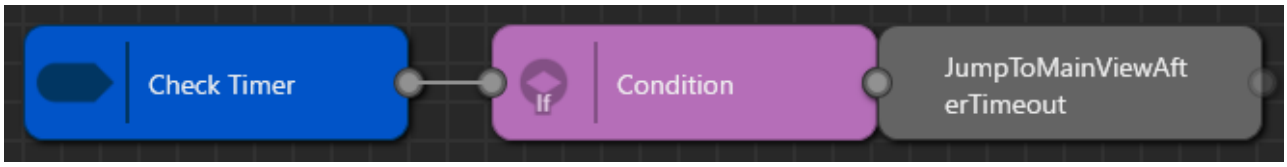
1. Double-click MainViewTree to open the View Tree panel.
2. Double-click DetailView to open the Action Chain Editor.
3. In the left pane, double-click ActionChain1 to open the diagram.
4. On the diagram, from the context menu Add Condition, select the Condition of the ReturnToMainOnTimeoutCondition you created earlier to add it.
5. From the context menu, select **Add Action > Action > Views > Jump To View** and place it on the diagram.
6. Select the Jump To View and configure the following properties in the Properties panel:

Property Category	Item	Value
General	Name	JumpToMainViewAfterTimeout
Asset	View	MainView

7. From the context menu, select **Add Trigger > Condition > Control > Check Timer** and place it on the diagram.
8. On the diagram, connect Check Timer to Condition.



9. Finally, snap the Jump To View action to the Condition to connect them.



This configuration means: when Check Timer fires, evaluate the ConditionOutput of ReturnToMainOnTimeoutCondition, and return to MainView only when the result is true. If you connect Check Timer directly to Jump To View, the system always returns after the timer elapses and cannot reflect the checkbox state. By placing a ConditionOutput in between, the transition becomes a conditional one.

Verifying in the Player

In this verification, you confirm not just the automatic transition by the timer itself, but also that the behavior changes depending on the checkbox state. This is the final check to verify that the division of roles between the Action Chain and the Condition is functioning correctly.

1. Save the solution and launch the Player.
   - Verify that MainView is displayed immediately after launch.

2. Click the Open Detail button.
   - Verify that the screen switches to DetailView.

3. Leave the Keep Detail Open checkbox off and wait for three seconds.
   - Verify that the screen returns to MainView automatically.

4. Click the Open Detail button again.

5. This time, turn the Keep Detail Open checkbox on.
   - Verify that the screen does not return automatically even after waiting about three seconds.

6. Verify that you can return to MainView by clicking the Back button.

In this page, you implemented a conditional automatic transition whose behavior changes according to the UI state, by coordinating the Action Chain Editor with the Condition Editor. With this, you have practiced all four basic features of the View Editor: creating screen-level Views in the View Definition Editor, configuring their relationships in the View Tree, building processing flows in the Action Chain Editor, and separating condition expressions in the Condition Editor. The complete flow of the tutorial is now finished.