Controls

This chapter gives an overview of what Controls are and shows how to use them.

Concept and Use Cases

Control Concept

Controls are template scenes created for later reuse via control nodes. They are meant to be interactive. To make them interactive they can be filled with Behaviors. A control collects a part of the scene graph and Behaviors into a single entity, which can be instantiated in another scene. All the controls are written in a special section which belongs to the asset library. When the solution is loaded from the asset, the content of the value of the control properties from the control node is set on the associated Behaviors in the control. Also the anchor nodes inside the user scene are added to the associated nodes in the control.

> Created controls will include Behaviors with the observation that the Behaviors will be included only if they are bound to a node from the created control.

> The interface of a control is defined by its control properties and anchors. The control properties are linked to the Behaviors inside controls and allow to change the value of the exposed properties. The anchors are linked to nodes inside controls and allow adding new children to those nodes.

Main differences of controls compared to regular scenes:

- Controls can be used in regular user scenes by dragging them into the scene like imported or Toolbox nodes
- Child nodes of a control (on any level) can be marked as anchors, if they are empty
- Controls have no cameras or lights
- Controls are meant to be interactive
- Controls can have public properties

Each scene node is uniquely identified by its parent using the type and the name. To identify a node globally its path to the scene should be used (for example /Scene:MyScene/Group:RootNode/Mesh:MyMesh).

The Scene Tree panel of a scene using a control will treat it as a single entity with all available anchor nodes as children. It is not possible to change the content of the control node, only the content of the anchors can be changed.

> The following impact has to be considered: Cross-references (nodes or behaviors) to different scenes may occur. For details see Cross-Reference Implications.

Control Use Cases

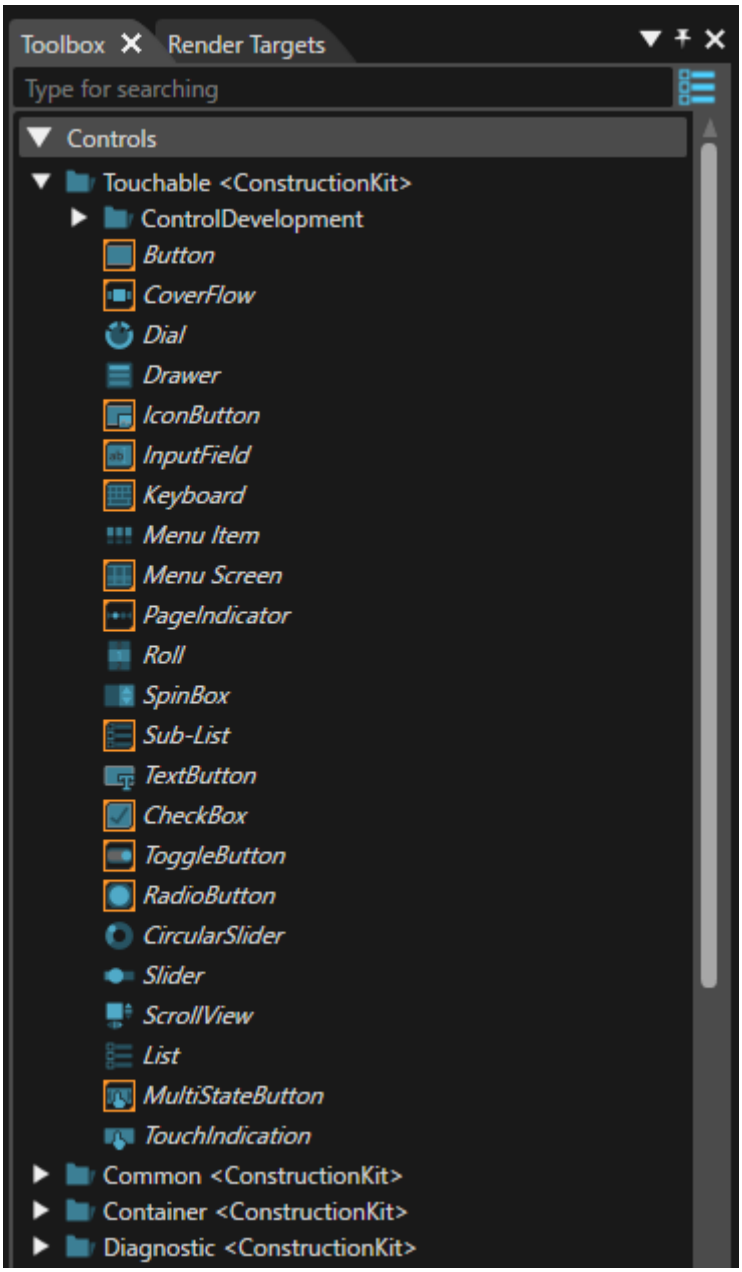
- Add new control content to all scenes: Imagine a control containing a car used in several scenes. To add a specific car accessory, simply link the accessory from the Imports panel to a proper anchor of the control. All scenes will be updated immediately.
- Add Behaviors to controls to make them interactive.
- Customize control nodes in specific scenes: In above car example, to use different car wheels in several scenes remove the initial meshes representing the wheels from the control and replace them with anchors to be customized in a scene.

> A conflict situation appears when two or more nodes have the same path (for example 2 billboards with the same name are added to a group). To avoid name conflicts when the controls are used, it is recommended to link anchors only to empty nodes and also to not link more than one anchor to a node.

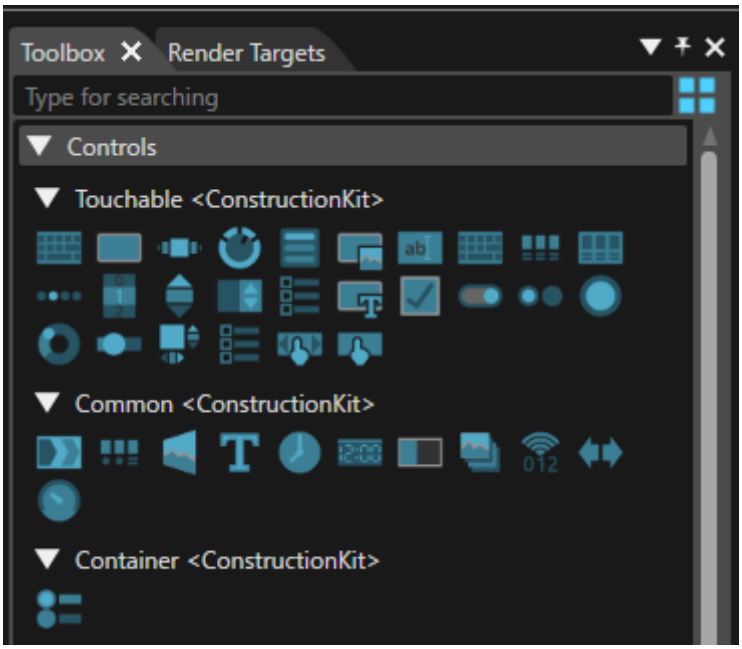
Use Controls

Use Controls

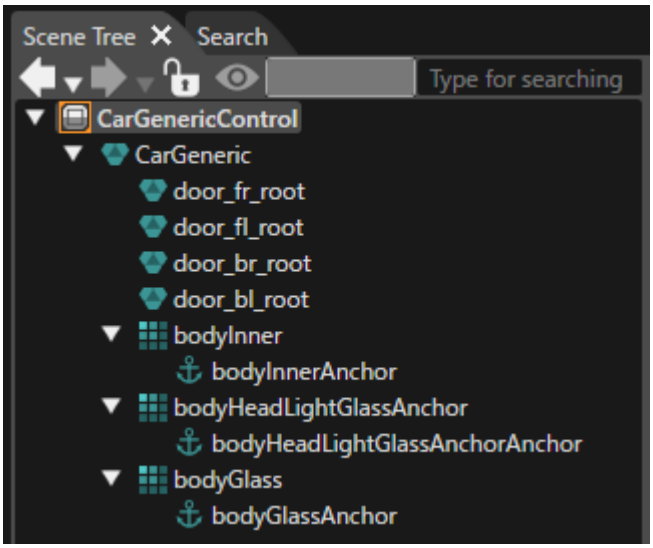
Controls are displayed in Toolbox panel from where they can be dropped into scenes which will result in control nodes.



A "Show Controls as Big Icons/List" button is available in the upper right corner. If this button is used, a different presentation mode - as icons instead of a text list of items - will be made available (see the image below).



The Scene Tree panel of a scene lists a control node as a single entity with all the anchors defined as children.

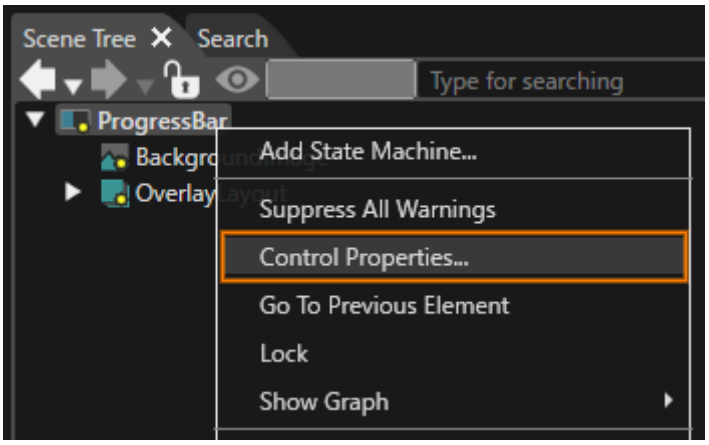


New content (other nodes) can be added from the Import or Toolbox panels to anchors by dropping a new node over an anchor and editing it in the Properties panel as usual.

Control Properties

By using the context menu the "Configure Public Control Properties" panel will get opened. This can be done by selecting the "Control Properties..." operation from the context menu.

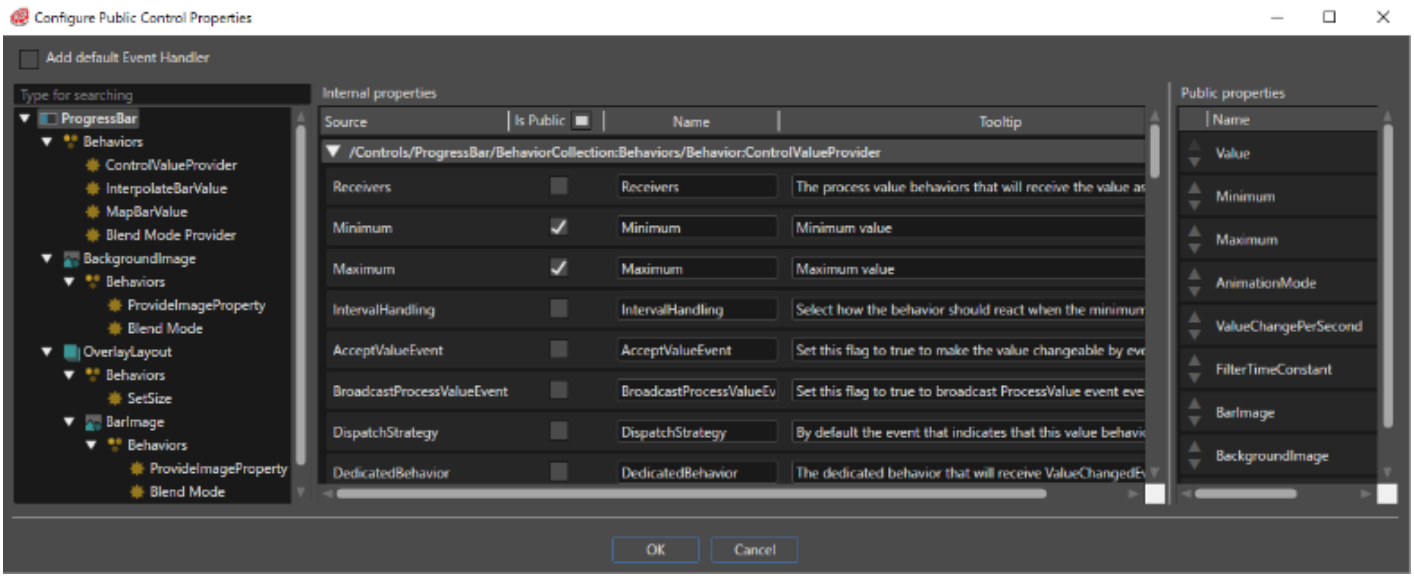
> The "Configure Public Control Properties" panel can be opened just when a control scene is selected - as can be seen in the image below.



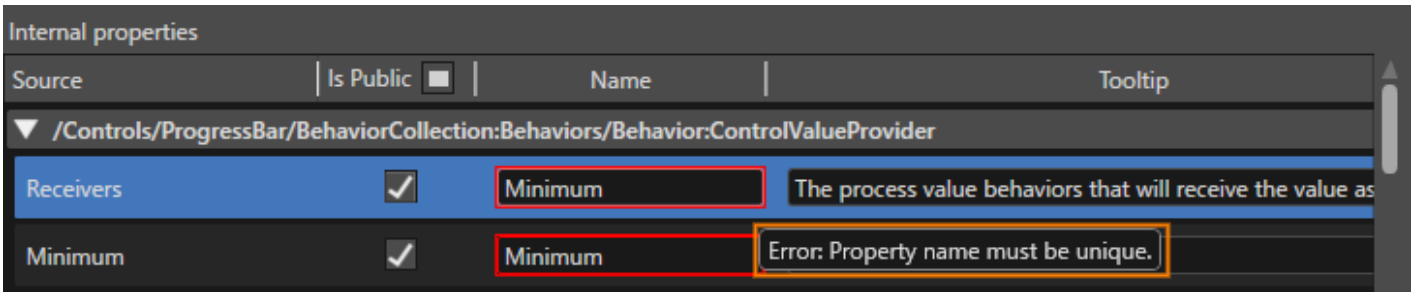
In this way, a new dialog "Configure Public Control Properties" opens where the user can choose which properties of the widget should be available (i.e. public) in the control.

> All the properties of those control nodes (i.e. instances of controls) which are present within a control scene - in conjunction with behaviors and widgets - are displayed in the "Configure Public Control Properties" dialog

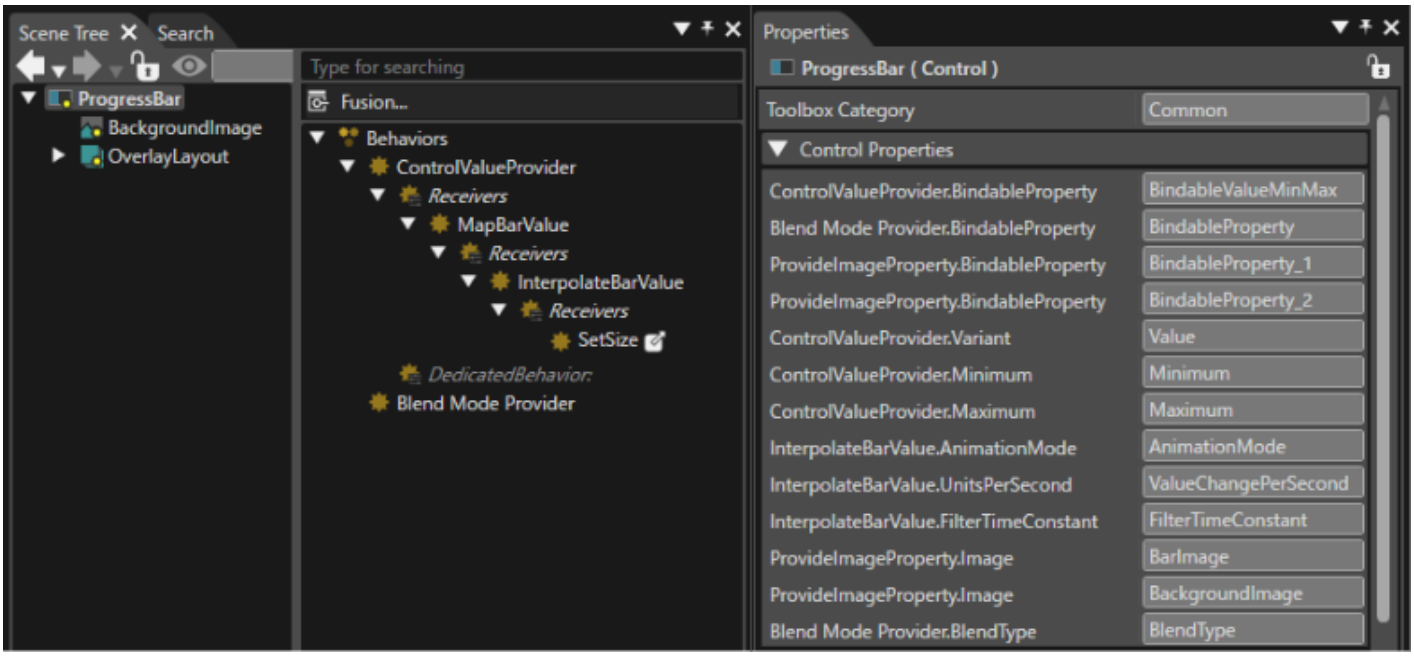
If necessary, the user can use the fields available under the "Name" and "Tooltip" sections to introduce any useful piece of information. The values filled by the user in the "Configure Public Control Properties" dialog will be visible in the "Properties" panel.



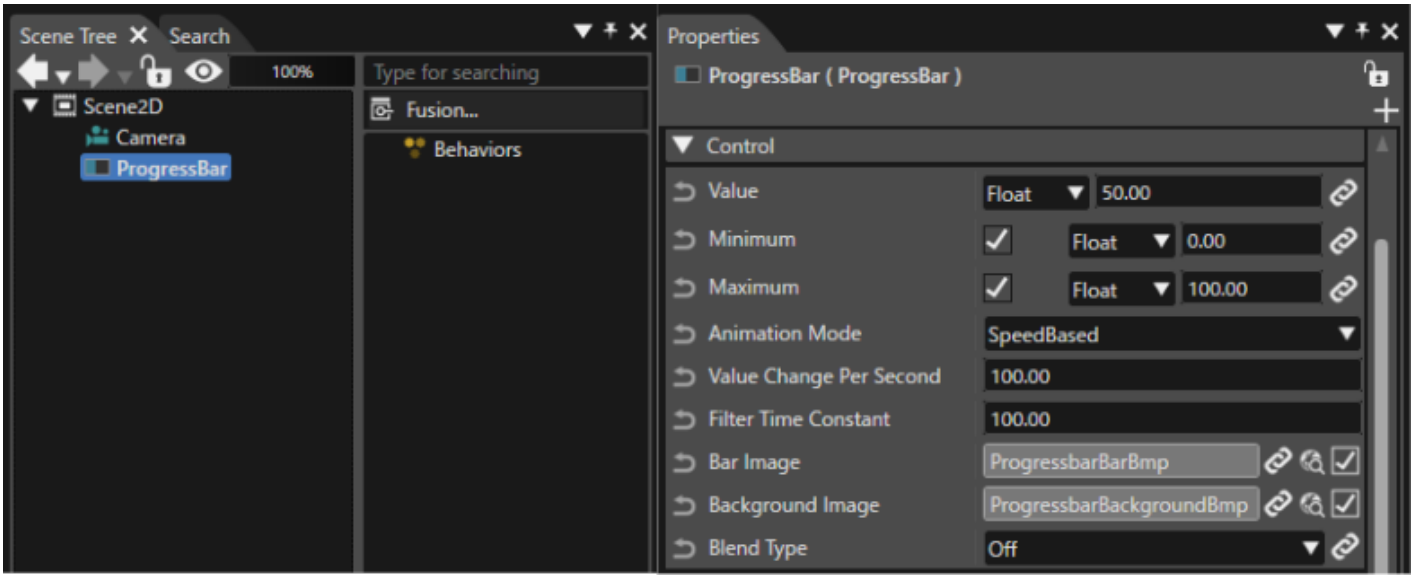
All the name of the properties have to be distinct/different. If not, a "Property name must be unique." warning message will be displayed.



If all the names are different, the properties which were selected as "public" will be displayed as public properties in the Properties panel. It is possible to remove any of these properties by using the "X" button.

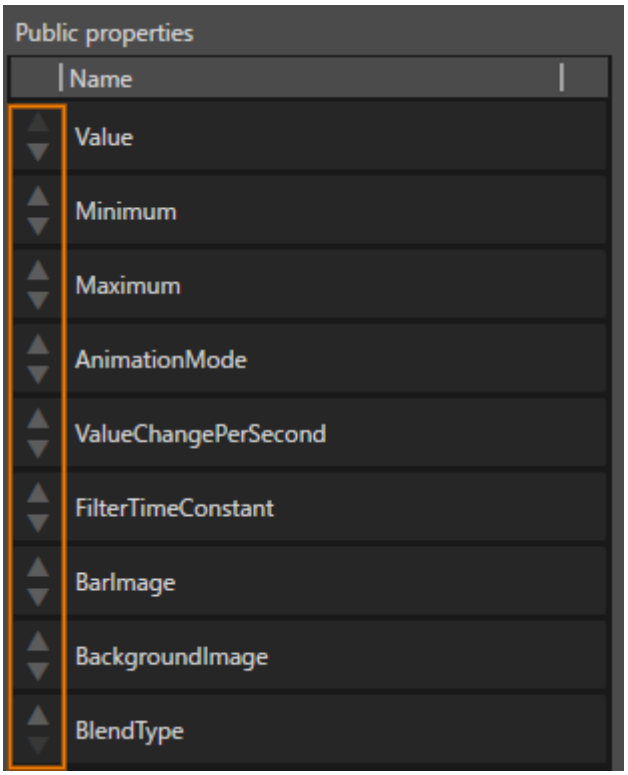


If the control with public properties is used in a scene, those properties will be visible in the Properties panel, too.



When the control is used in a scene, any of the public properties can be set or modified but none of the properties can be removed.

The display order of public properties in the property panel can be set in the [Public Control Property Configuration] dialog. You can change the display order by clicking the button to the left of each property name displayed in the Public properties, or by dragging and dropping each property.



Control Updates

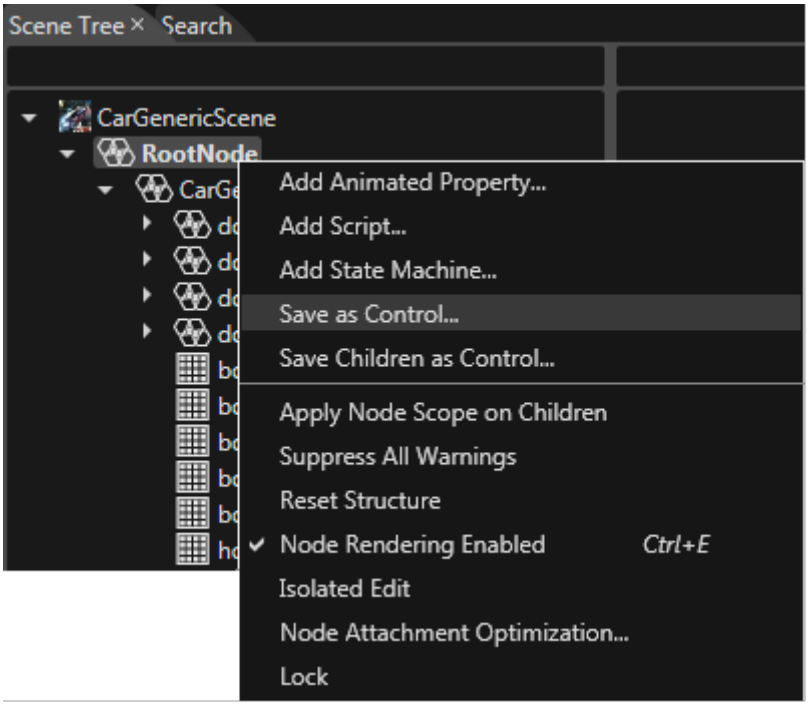
When a control is being edited, the changes are reflected in all the scenes using that control.

Create Controls

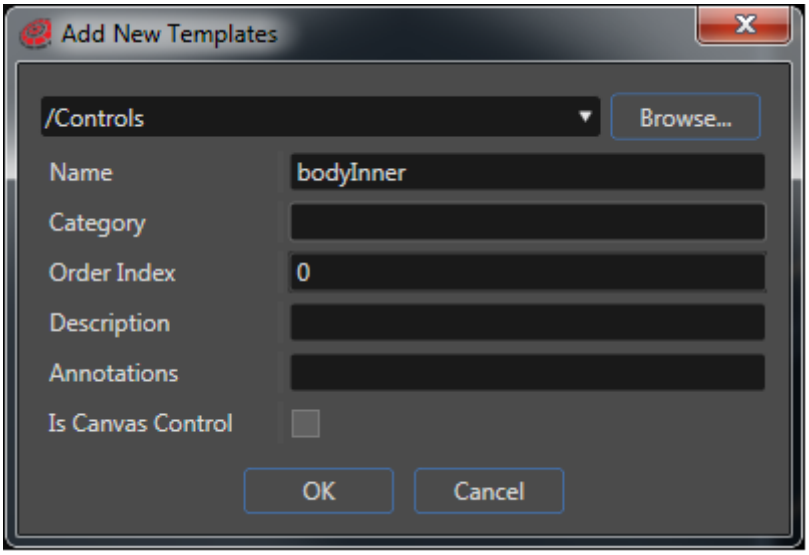
Save a Node As Control

Save nodes as control using:

- "Save as Control" context menu item on a selected node in the Scene Tree panel. The new control will contain both node and all its children.
- "Save Children as Control" context menu item on a selected node to create the control only from the node children. In this case, the control won't inherit the parent node's actual position, but will have default transformation properties instead.



In the "Add New Template" dialog, a location within the solution and a name for the new control is defined.



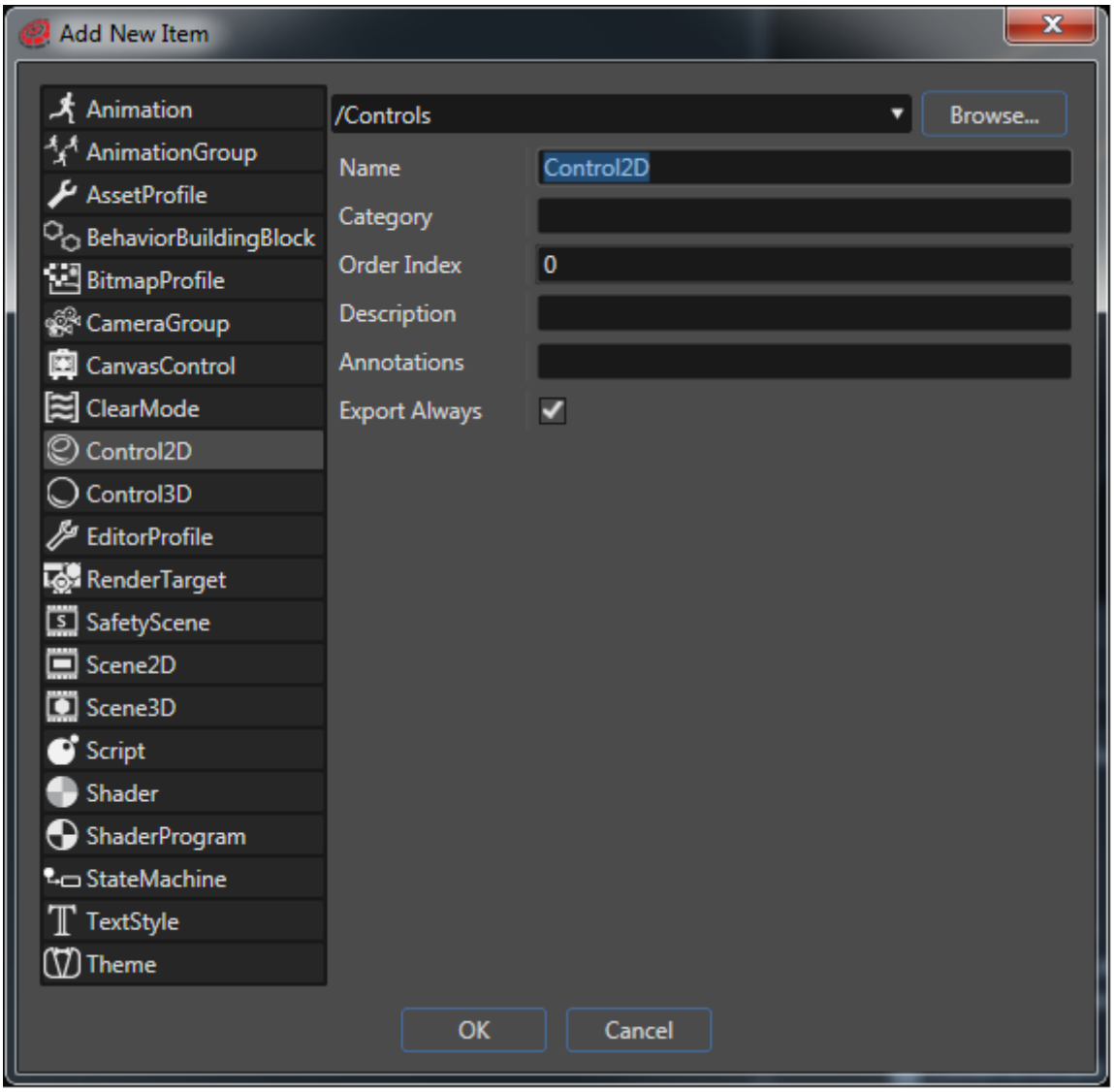
The new control will be available in the

- Solution Explorer panel, and in the
- Toolbox -> Controls.

Open the control for editing with a double click.

Create an Empty Control

In Solution Explorer, use the context menu "Add New Item" to create a new control.



Edit Controls

Edit Controls

Controls are listed in Toolbox-Controls panel and Solution Explorer panel. By double clicking on a control, it will be opened for editing just like a normal scene.

Editing controls allows to:

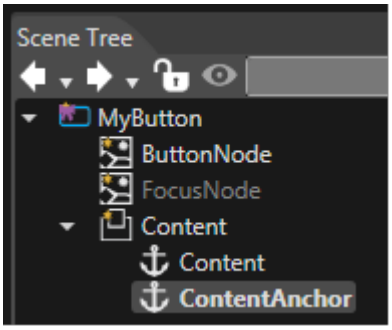
- Modify the scene graph structure and content of the control;
- Create and modify anchors to a control.

It is possible to set a description for the control properties. This description will be shown as tooltip in the properties panel for the associated property of the the control node.

Create Anchors

The Toolbox panel provides an "Anchor" node in addition to regular scene toolbox items but no cameras, when editing controls. An anchor will later be used by a scene to attach further content to the control.

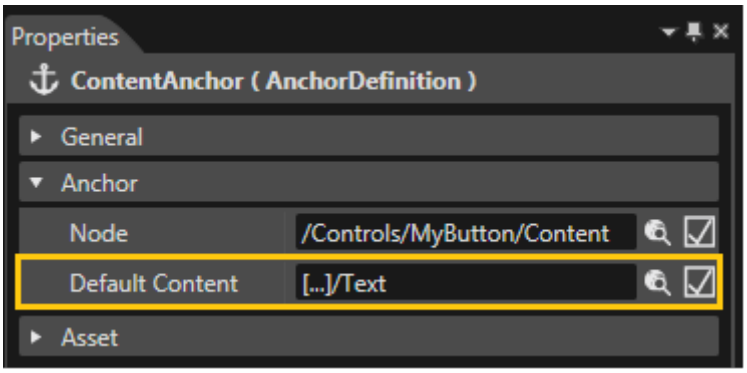
Add an anchor dragging it from the Toolbox and drop it on a desired node part of the control. The anchor will appear as a new child of the control in the Scene Tree panel, named after the parent node on which it is dropped:



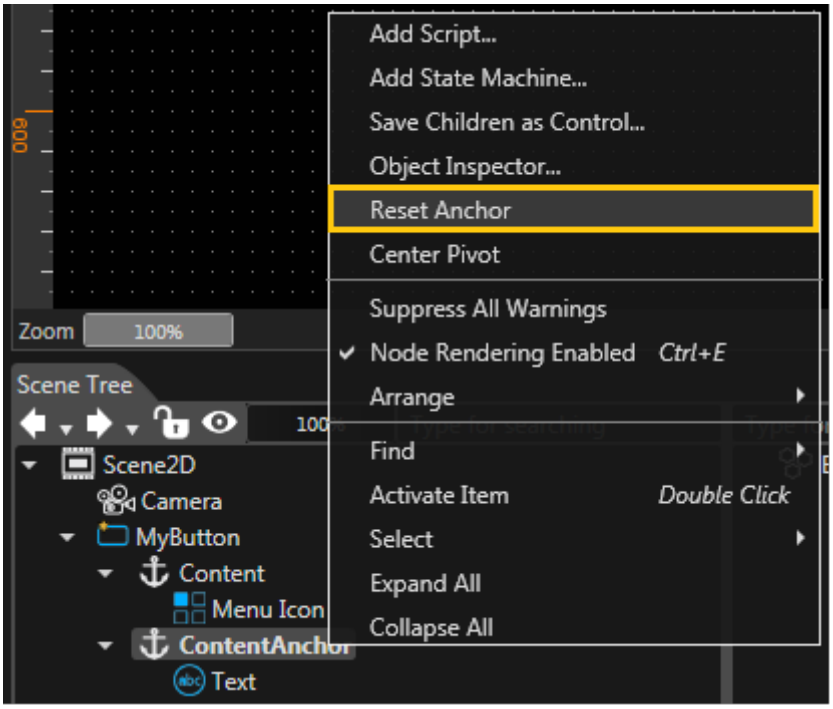
The node an anchor is assigned to can be modified in the Properties panel of the anchor.

Further, all anchors of the control are also listed in the Properties panel of the control.

The control functionality provides support to add default content to anchors. When a control is instantiated in a scene, in the anchors where the "Default content" is specified, a control node of that type will be instantiated and added as child to the anchor node itself.

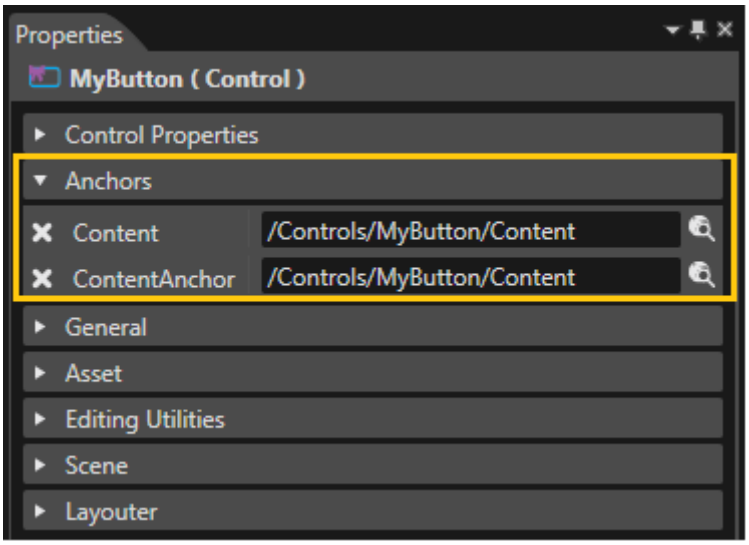


A specific option is available in the context menu of a control instance which has an anchor with default content. The name of this option is "Reset Anchor". By using this option it is possible to reset anchors to the default content.



Delete Anchors

Delete an anchor in the Scene Tree panel via context menu or in the Properties panel of the control by using the "X" button on the left side of the anchor:


Properties
MyButton (Control)
Control Properties
Anchors
Content
/Controls/MyButton/Content
ContentAnchor
/Controls/MyButton/Content
General
Asset
Editing Utilities
Scene
Layouter

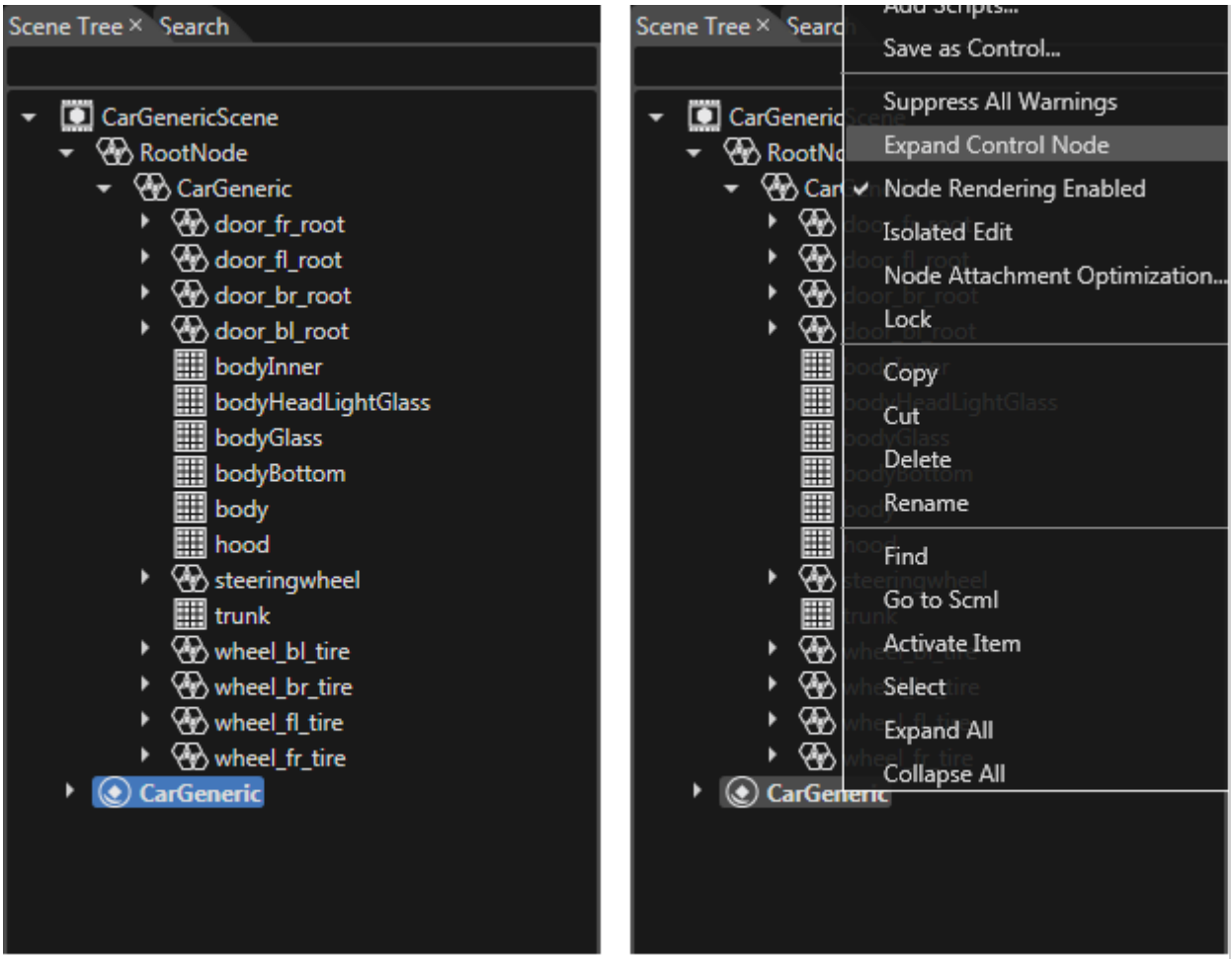
Expand Controls

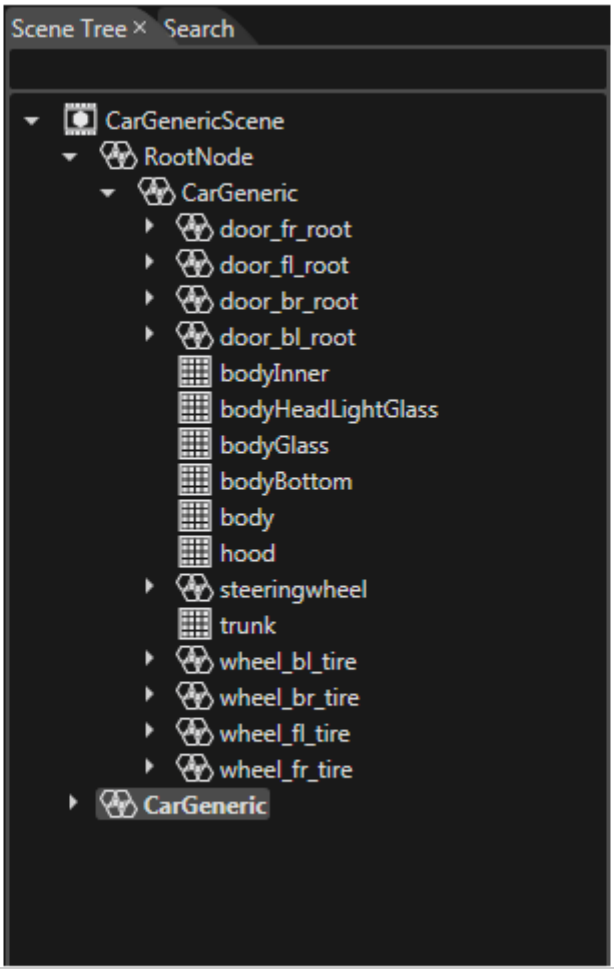
Expand Controls

If a control node needs to be customized for a specific set of scenes, it might be useful to decompose an existing control to create a new, customized version of it.

For this, use the context menu option "Expand Control Node" on a selected control.

In this case, the control node will lose the link to the original control source and will become a usual node; any modification of the control will not affect it anymore.


Scene Tree
Search
CarGenericScene
RootNode
CarGeneric
door_fr_root
door_fl_root
door_br_root
door_bl_root
bodyInner
bodyHeadLightGlass
bodyGlass
bodyBottom
body
hood
steeringwheel
trunk
wheel_bl_tire
wheel_br_tire
wheel_fl_tire
wheel_fr_tire
CarGeneric
Save as Control...
Suppress All Warnings
Expand Control Node
Node Rendering Enabled
Isolated Edit
Node Attachment Optimization...
Lock
Copy
Cut
Delete
Rename
Find
Go to Scml
Activate Item
Select
Expand All
Collapse All


Scene Tree
Search
CarGenericScene
RootNode
CarGeneric
door_fr_root
door_fl_root
door_br_root
door_bl_root
bodyInner
bodyHeadLightGlass
bodyGlass
bodyBottom
body
hood
steeringwheel
trunk
wheel_bl_tire
wheel_br_tire
wheel_fl_tire
wheel_fr_tire
CarGeneric

Augmented Reality Controls (AR Controls)

1. Introduction

This section describes CGI Studio controls to enable augmented reality applications with CGIStudio, which were designed for generic navigation overlay use case in the automotive area. Typically in **video see-through** AR, a real world video feed is enhanced with graphical elements.

---

2. Use Case: Navigation Overlay

Major elements for this use case are:

- **Video Streaming:** Rendering from a video source into a texture.
- **Camera Tracking:** Updating the moving AR camera via data binding.
- **Navigation Path:** Visualization of the proposed path a navigation system.
- **Points of Interests (POIs):** Loading POIs (e.g. street names, buildings, ...) from an external dictionary and place them in the scene at runtime. Metadata is received as JSON formated string.

Detailed descriptions can be found in Predefined Controls and Predefined Behaviors.

---

3. Prerequisites

3.1. Cmake flags

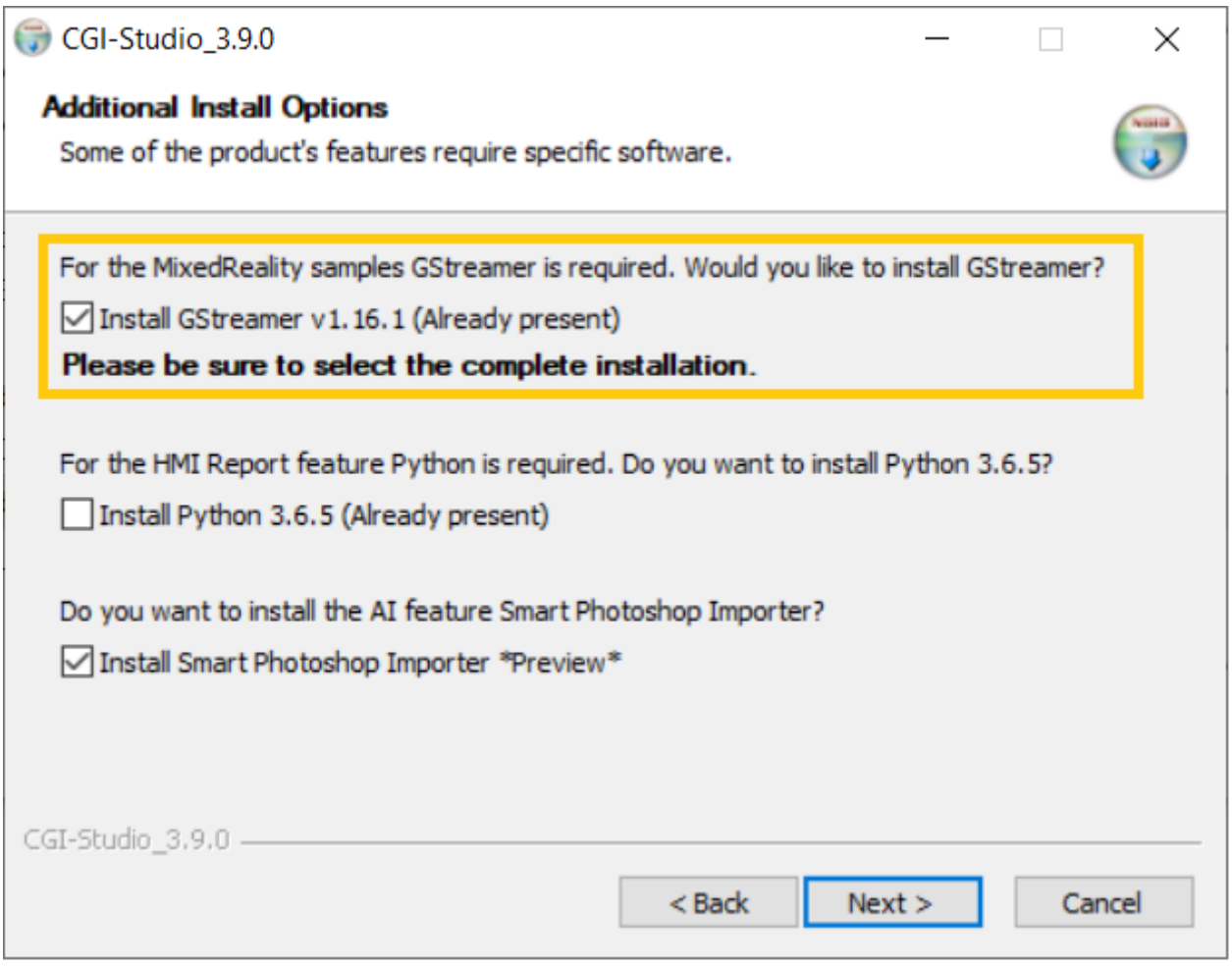
Enable and disabling mixed reality and streaming features:

- `CGIAPP_SAMPLE_STREAMING_ENABLED` is used for the streaming feature. It includes GStreamer libraries from the 3rd party directory.
- `CGIAPP_SAMPLE_MIXEDREALITY_ENABLED` is used for all augmented reality related behaviors and controls.

3.2. GStreamer

GStreamer is the default implementation of the VideoStreamer interface. It is used to decode the video source and provides the data to the VideoStream Behavior. If you want to use GStreamer, a working installation of GStreamer is required. The GStreamer version provided in the CGI Studio 3PSW is GStreamer version 1.16.1. Successfull installation of GStreamer modules can be verified with the console commmand `gst-inspect-1.0`.
Note that, under Windows you might need to add GStreamer to the environment variables.
Instructions can be found here: https://gstreamer.freedesktop.org/documentation/installing/on-windows.html

Please also note that, the Android Evaluation Package comes with an installer and by default, the CGI Player is built with streaming enabled. The recommendation is to check the "Install GStreamer" option in the provided installer, and do a "Repair" installation regardless of whether GStreamer is installed or not, because of some dependencies.

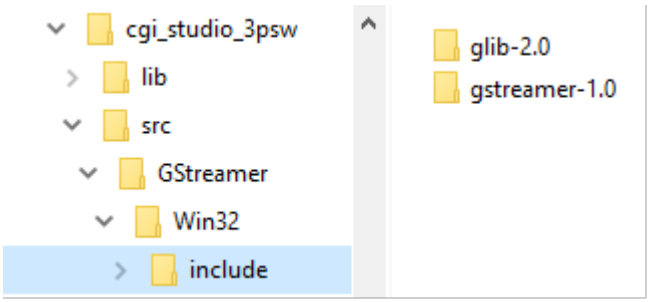
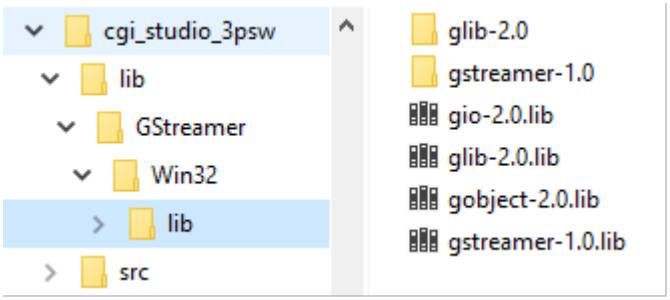


If in any case the CGI Player still looks for a missing dll, check that the GStreamer is set in the PATH environment variable, or copy the GStreamer dlls to the same location as the CGI Player executable.

For more general information, please visit the official GStreamer website: https://gstreamer.freedesktop.org/.

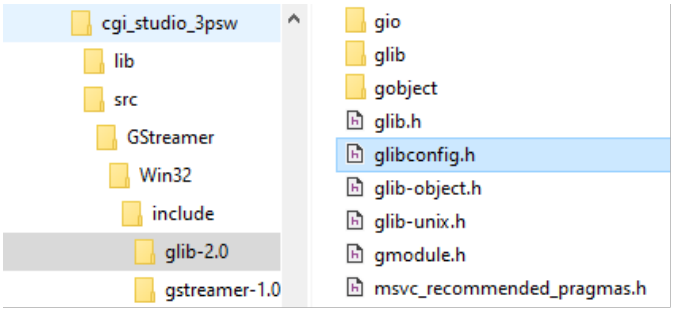
3.3. How to build the CGI Player using GStreamer and Microsoft Visual Studio

- Make sure you have a working Microsoft Visual Studio installation.
- Close Microsoft Visual Studio.
- Install GStreamer for Windows developer and runtime:
  - https://gstreamer.freedesktop.org/data/pkg/windows/1.22.12/mingw/gstreamer-1.0-devel-mingw-x86_64-1.22.12.msi
  - https://gstreamer.freedesktop.org/data/pkg/windows/1.22.12/mingw/gstreamer-1.0-mingw-x86_64-1.22.12.msi
- If you do not have GStreamer directories in the folder `cgi_studio_3psw/lib` and `cgi_studio_3psw/src` you need to copy the following files and folders from your GStreamer installation directory:

Copy the following folders from your GStreamer installation directory (`C:/gstreamer/1.0/msvc_x86/include`) to folder `cgi_studio_3psw/src/GStreamer/Win32/include` • glib-2.0 • gstreamer-1.0	
Copy the following folders from your GStreamer installation folder (`C:/gstreamer/1.0/msvc_x86/lib`) to folder `cgi_studio_3psw/lib/GStreamer/Win32/lib` • glib-2.0 • gstreamer-1.0 Copy the following library files from your GStreamer installation folder (`C:/gstreamer/1.0/msvc_x86/lib`) to folder `cgi_studio_3psw/lib/GStreamer/Win32/lib` • gio-2.0.lib • glib-2.0.lib • gobjext-2.0.lib • gstreamer-1.0.lib	

Copy the following file from your GStreamer installation folder (`C:/gstreamer/1.0/msvc_x86/lib/glib-2.0/include/glibconfig.h`) into folder `cgi_studio_3psw/src/GStreamer/Win32/include/glib-2.0`

- glibconfig.h



- Open CMake
- Input the location of your source code into the first input field "Where is the source code:" (*<cgi-studio-root>/cmake/Candera/Player*).
- Input the location where to build the binaries to into the input field "Where to build the binaries:"
- To be sure you have a clean build, go to the menu "File - Delete Cache".
- Click the "Configure" button in the middle of the CMake window on the left side.
- A dialog will be opened, choose "Optional platform for generator": Win32
- Wait for CMake to finish configuration.
- In the CMake defines, adjust the following settings:
  - change COURIER_PLATFORM to "imx6platform"
  - enable CGIAPP_SAMPLE_STREAMING_ENABLED
  - enable CGIAPP_SAMPLE_MIXEDREALITY_ENABLED
- Click the "Configure" button again.
- Wait for CMake to finish configuration.
- Press the "Generate" button (just right to the "Configure" button).
- Wait for the Microsoft Visual Studio Solution to be generated.
- Navigate to the Microsoft Visual Studio Solution file in the folder "Where to build the binaries".
- Open the Solution file "Player.sln" with your working Microsoft Visual Studio.
- Wait for the project to load.
- In the Microsoft Visual Studio menu press "Build - Build Solution".
- Wait for the solution to be built.

Finally, you can start the Player application.

---

4. Interfaces

4.1. Events

- **VideoStateEvent:** Event emitted from Video Stream Behaviors to registered behaviors when video state changes. In our sample, the video stream control uses this event to let the PoseTracking behavior know that the video has started and therefore the camera tracking should be started.

- **ArTargetEvent:** Event emitted from target nodes to tracker components to start or stop tracking. In our sample, the PoseTracking Behavior is sending an AddTarget event as message to the CameraTrackerComponent to notify it, that the tracking should start.
- **TrackingEvent:** Event emitted from PoseTracking Behaviors to registered behaviors when the tracking state changes, the position or rotation are updated by the tracker. Possible tracking states are: *Registered, Detected, Tracked,* and *Lost*. In our sample, the PoseTracking Behavior on the node that holds the camera emits this event to the PoiManager Control and the NavigationPath Control, such that they are also informed about the current camera position and rotation, and the current tracking state.
- **PoiUpdateEvent:** Event sent from PoiManager to one of its POIs when a new metadata update arrives. A PoiBehavior will handle the event accordingly. It has a PoiUpdateEventType that states which kind of update this event represents, and a Variant with the value. In our sample, the PoiManager is creating these events based on JSON metadata messages and sending them the respective PoiBehaviors.

4.2. Messages

- **ArTargetEventMsg:** Used to send the ArTargetEvent as message from the view to the model. The only member of the message is the event.
- **PoiReqMsg:** Used by the PoiManager to request a new list of points of interest from the model. Members are the start and end position, which the model uses to build a bounding rectangle. It will then search for POIs that lie within the bounds of this rectangle and put them into the PoiResMsg.
- **PoiResMsg:**Reply to the PoiReqMsg. Holds a vector of PoiData.

---

5. Known Issues and Limitations

5.1. Streaming

5.1.1 Video Stream

- The property "FlipVertically" in the VideoStream does only work in 2D VideoStream control. In 3D one has to change the UV texture mapping to flip the image. This is why the "no signal" texture is displayed upside down in the 3D sample scenes.

5.1.2 GStreamer

- GStreamer only supports Windows and Linux platforms.
- The default GStreamer modules only support the "Baseline" profile of the H.264 codec.

5.2. Mixed Reality

5.2.1. PoseTracking

- Camera position is a FeatStd::Vector3, which consists of three `FeatStd::Float` values. Therefore, there is a limit on how far the camera can move. Could be an issue.
- Static offsets to position and rotation of the camera that might result from mounting the tracker to the camera must be applied to the camera directly. Offsets cannot be set via data binding so far.

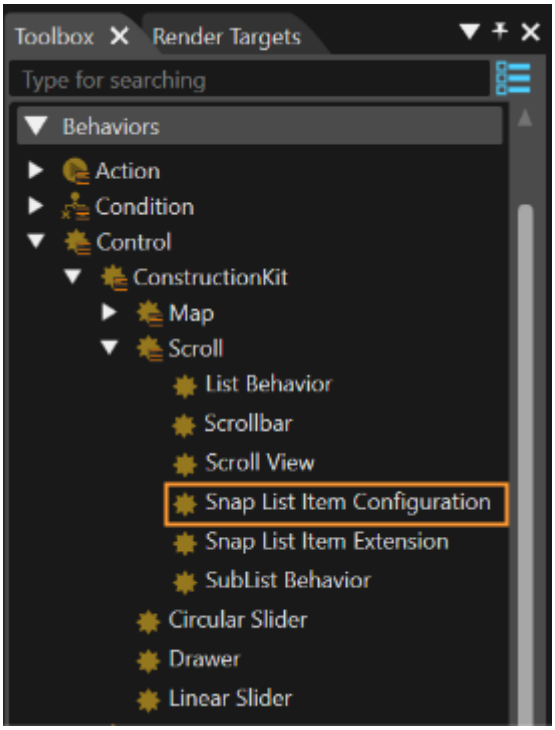
5.2.2. Points of Interest

- Position is not part of metadata message, but independent. It would probably make sense to just move it to metadata and let the respective PoiBehavior position itself in space. The PoiManager does not have to be the one to do it. In fact, the PoiManager already forwards the camera position to the PoiBehavior, but so far, the PoiBehavior only uses it to handle the level of detail fading.
- Alignment is handled by the PoiManger, as in early design, PoiManager was the only one who had access to the camera position. However, this might as well be forwarded to the PoiBehavior as part of the metadata.

List Control - Snapping Mechanism

Snap List Item Extension

The **Snap List Item Extension Behavior** is a Control > Scroll Behavior that allows to configure a snapping mechanism for a List Control. To use this snapping mechanism, drag and drop a Snap List Item Extension behavior onto a **List control** in your Scene Editor or Scene Tree panel.



The **Snap List Item Extension** behavior considers the following use cases:

- When a new item is focused, the list snaps to this item at the aligned snapping position of this item with a snapping animation.
- When the list is scrolled by direct touch, scroll of the list or by external scroll via scrollbar, the list will snap to the item with the smallest distance to the aligned snapping position.
- When the list is scrolled by a next/previous page scrollable event the list will snap to the item with the smallest distance for that new page to the aligned snapping position.
- When the list is scrolled by a next/previous item scrollable event, the list will snap to the newly requested item to the aligned snapping position.
- When a list is flicked by a touch gesture, the final target position will be adjusted in a way that the list snaps to the item with the smallest distance at the flick target position that is determined by the flick speed.

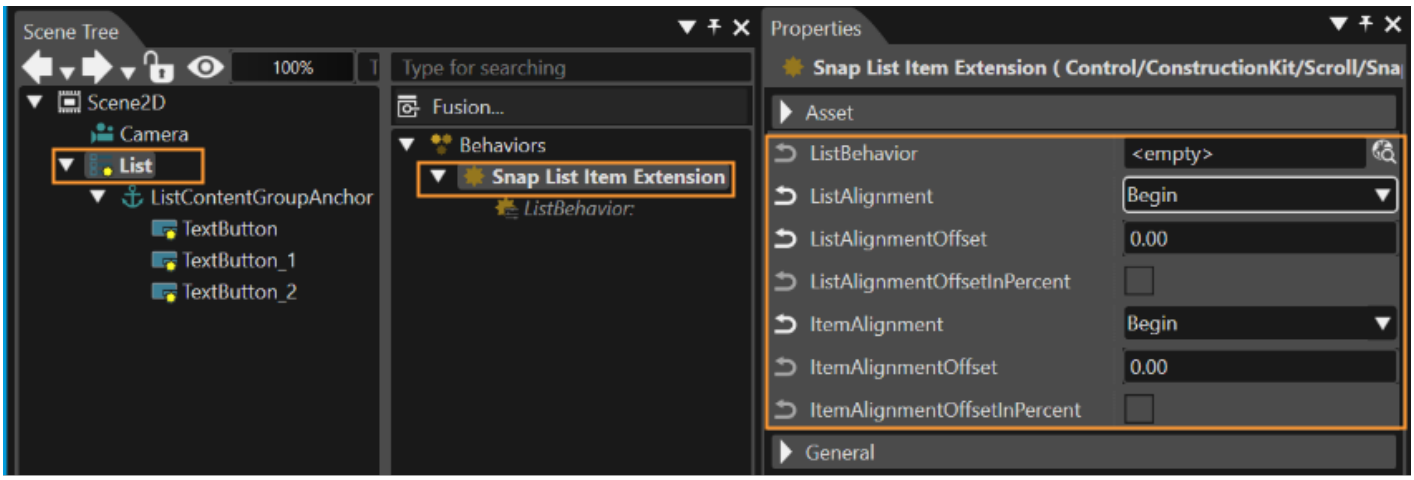
This snapping mechanism supports

- static and dynamic lists
- vertical and horizontal lists
- left-to-right and right-to-left direction
- the wraparound feature for static lists
- restoring the last focused index
- disabled items (Note: not supported for dynamic lists)

> Wraparound is only available for static lists with an accumulated complete item size that is larger than the arrange area of the list.

Configuration

The **Snap List Item Extension** behavior can be applied to a list control via drag and drop from the toolbox.



In case you are using a custom ListControl, it is recommended to configure the ListBehavior of the ListControl as the ListBehavior of the SnapListItemExtension behavior. Please see Configure ListBehavior in Custom List for more information.

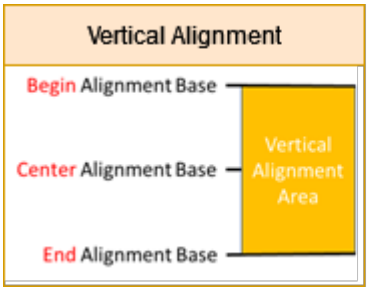
The properties *List Alignment* and *Item Alignment* define the snapping position. The snapping position can be refined by the *List Alignment Offset* and *Item Alignment Offset* properties. The properties *List Alignment Offset In Percent* and *Item Alignment Offset In Percent* define if the unit of the offset is in *percent* or in *pixel*.

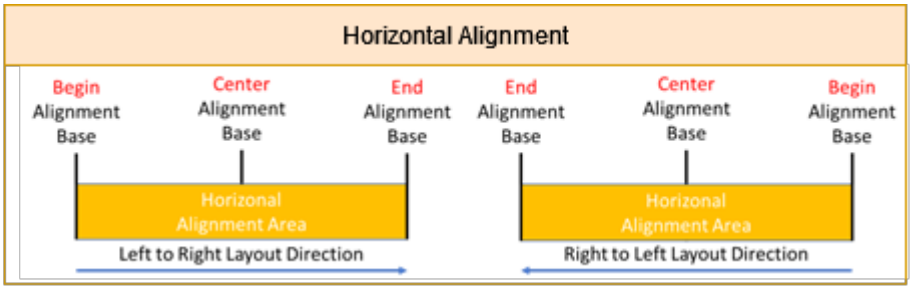
List/Item Alignment

These properties can be set to

- Begin
- Center
- End

The base alignment for item/list is defined as follows:



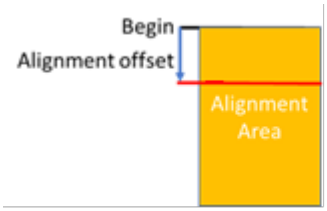


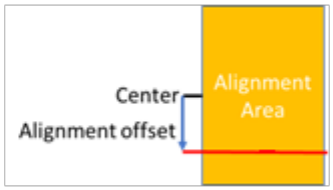
List/Item Alignment Offset

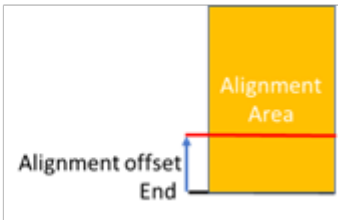
The Alignment Offset properties allow to refine the snapping position.

The following rules apply to the alignment offset properties:

- For **begin** and **center** alignment the offset is directed towards the end of the arrange area.
- For **end** alignment the offset is directed toward the begin of the arrange area.
- In case of **percental offset configuration**, the offset value has to be **between 0 and 1** and will be multiplied with the alignment area size to calculate the absolute offset.

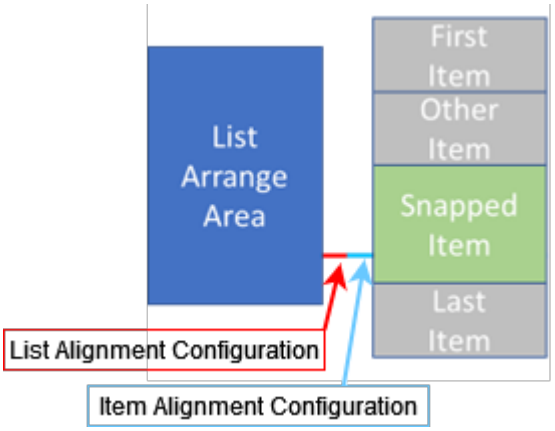




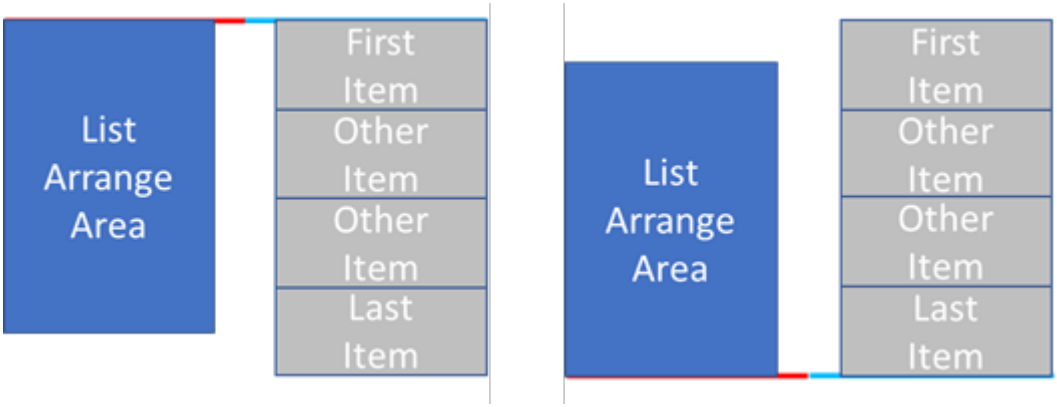


Aligning the List and the Snapped Item

The configuration in the properties describe an alignment point for the list and an alignment point for the snapped item. The alignment point of the list and the alignment point of the item shall be aligned at the same visual position for a snapped item.



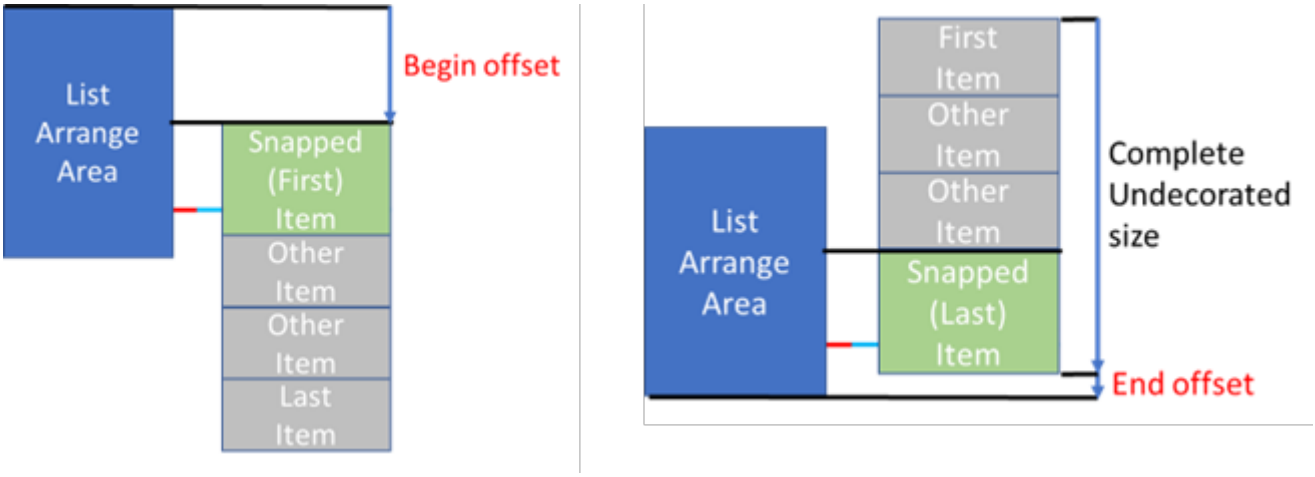
For a normal *ListBehavior instance* the item alignment starts with the first item presented on top of the list arrange area and stops with the last item at the bottom list arrange area:



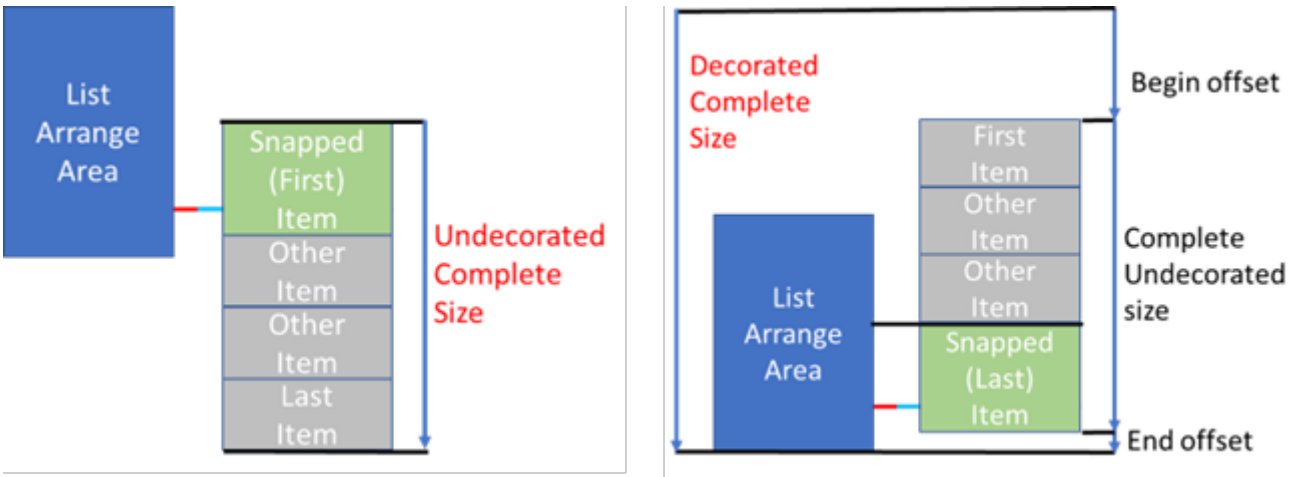
So, the accumulated size of all items represents the complete size of the virtual area presented by the list that can be accessed via scrollbar and scrollable events.

Due to the static item snap position the list

has to present additional empty space before the first item as a begin offset and also after the last item as an end offset:

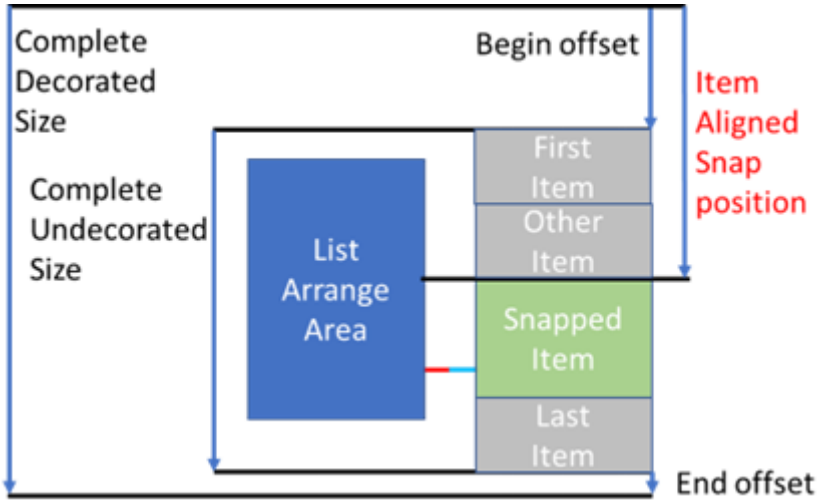


So, the normal list without snapping extension presents an undecorated complete size of its items. But the snapping extension has to add the begin and end offset to this undecorated complete size to present the decorated complete size as complete size of the scrollable interface:



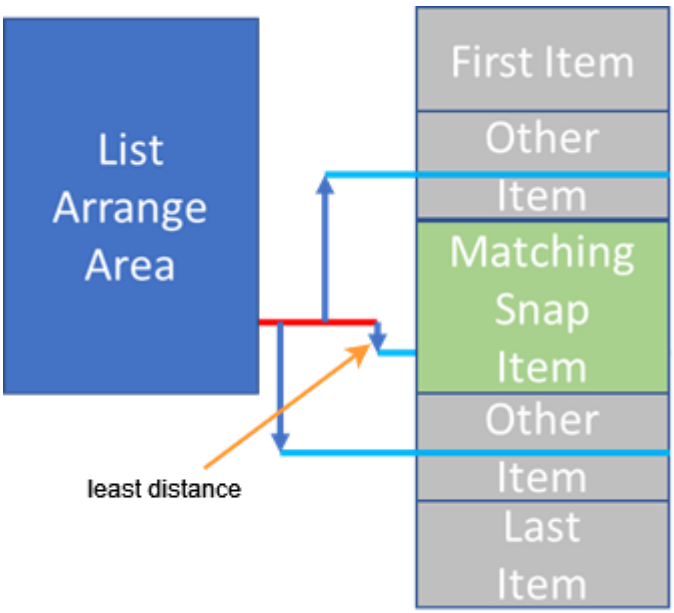
This begin offset also affects the position of the item. So, the first item is virtually positioned at the virtual position of the begin offset. The final aligned snap position of an item has to reflect the

begin offset, the accumulated size of all predecessor items, the item alignment offset and the list alignment offset.



Find the Matching Snap Item

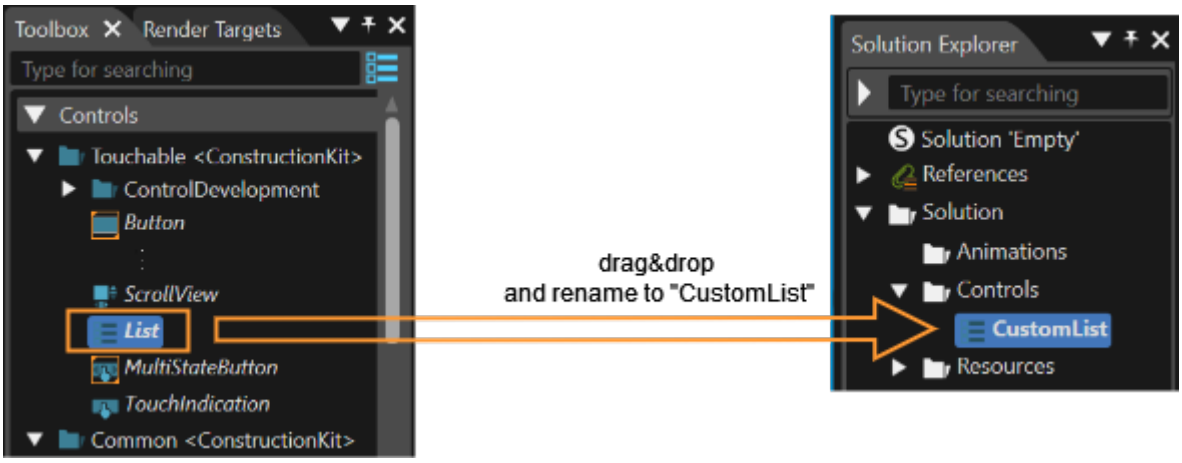
Whenever the position of the items in the list is changed (e.g. when scrolling or flicking or the like) the snapping extension will determine the item to snap to. The item, whose alignment position has the *least distance* to the list alignment position, will be the matching snap item.



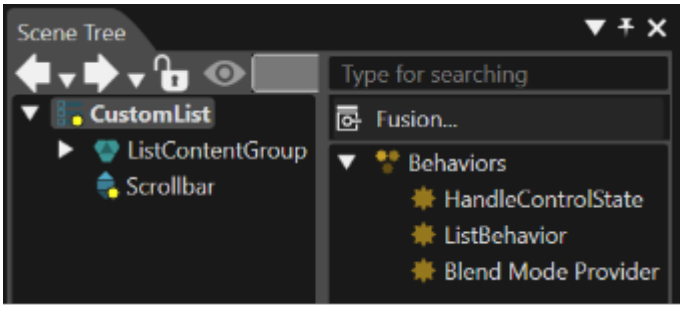
Configure ListBehavior in Custom List

The *Snap List Item Extension*'s ListBehavior can be configured in the custom List Control.

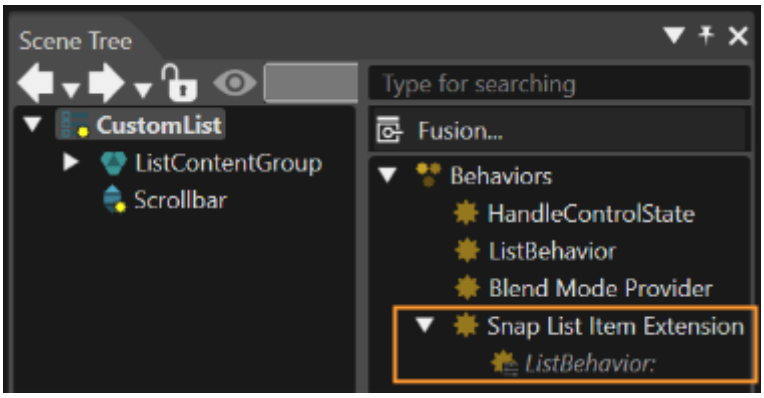
Open your custom list control with a double click or create your custom list control by dragging the predefined list control from the Toolbox into the Controls folder of your solution in the Solutions Explorer.



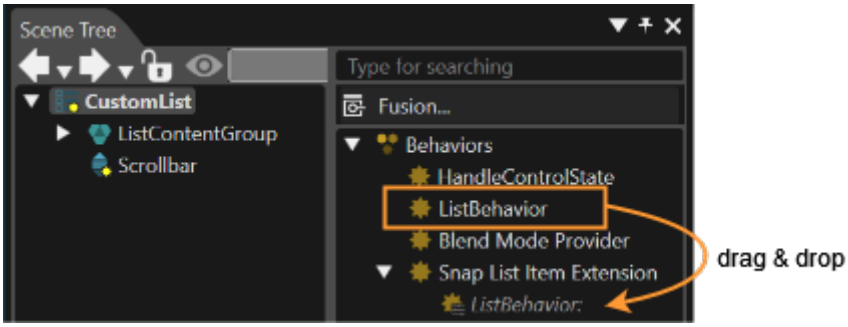
The CustomList looks like this in the Scene Tree:



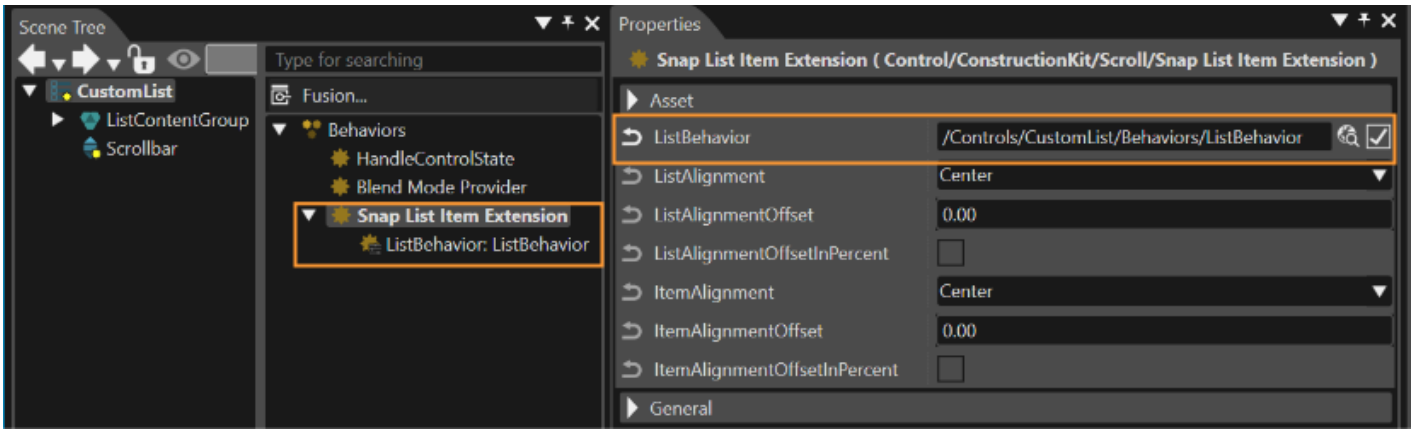
Drag and drop a Snap List Item Extension behavior to the CustomList:



Drag the ListBehavior from the CustomList's Behaviors down to the *SnapListItemBehavior*'s ListBehavior:



Now you can see that the ListBehavior property has received a new value:

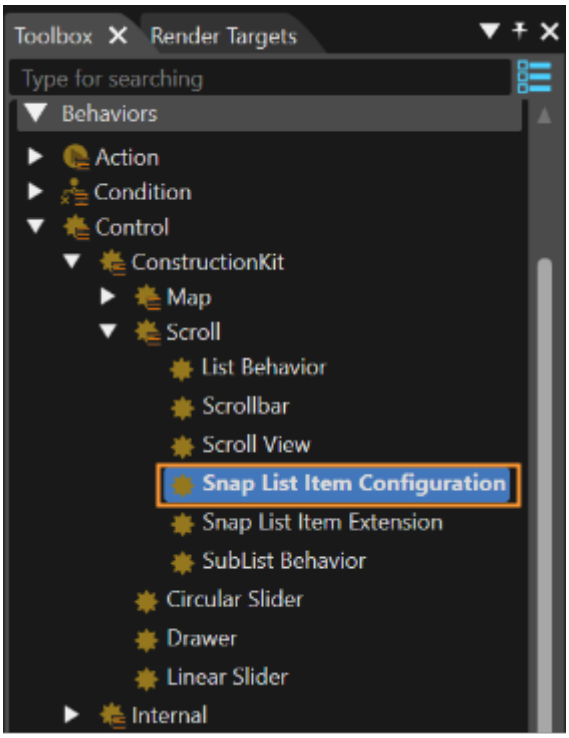


Snap List Item Configuration

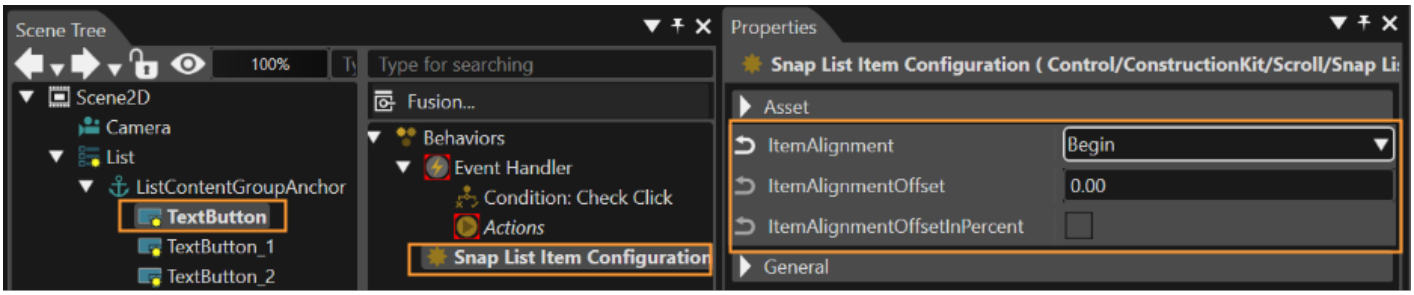
The alignment position of a list's items is configured in the *Snap List Item Extension* behavior with the following properties:

- ItemAlignment
- ItemAlignmentOffset
- ItemAlignmentOffsetInPercent

In case certain items should be configured with a different alignment position, you can apply a **Snap List Item Configuration** behavior to those items that should be handled differently.



Drag and drop this behavior from the Toolbox onto the desired list item to be able to configure a different alignment position for this item.


Scene Tree
100%
Type for searching
Scene2D
Camera
List
ListContentGroupAnchor
TextButton
TextButton_1
TextButton_2
Fusion...
Behaviors
Event Handler
Condition: Check Click
Actions
Snap List Item Configuration
Properties
Snap List Item Configuration (Control/ConstructionKit/Scroll/Snap Li
Asset
ItemAlignment
Begin
ItemAlignmentOffset
0.00
ItemAlignmentOffsetInPercent
General

Breadcrumb - StateMachine Workflow

Breadcrumb is not a standalone control, it does not trigger internally any scene transitions so it should be always connected with external state machine. For that purpose, breadcrumb property *State Machine Node* should be set to a node that contains a *State Machine* behavior.

Data flow for *SetPath*:

- Scene element (for example, button).
- State machine has a defined transition that reacts to the click of a button that was defined in the previous point.
- State machine has defined states that contain actions for entry (e.g. request for scene transition, breadcrumb action with 'SetPath' option).
- After click, State machine transitions to defined state which triggers breadcrumb action.
- Breadcrumb control visualizes proper data which received from path.

Data flow for *BackToPrevious* action:

- Scene element (for example, button). It cannot be a breadcrumb node as a child.
- User defines how the action will be called e.g. OnClick
- User connects breadcrumb action to action slot.
- After calling this action, breadcrumb sends a new path to state machine node.

Data flow for *BackToElement* action:

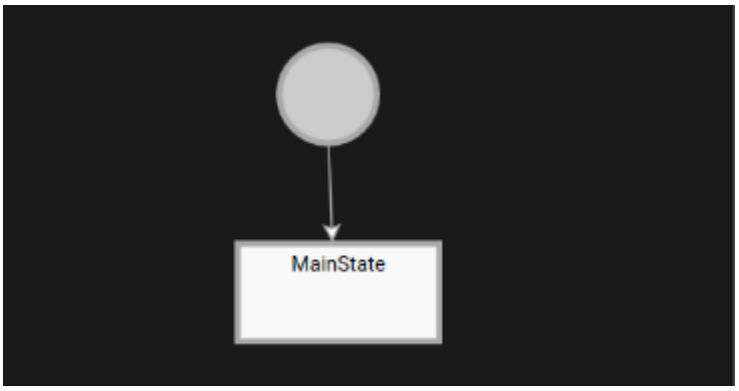
- Same as above but the state machine should contain the related transitions.

State Machine Example

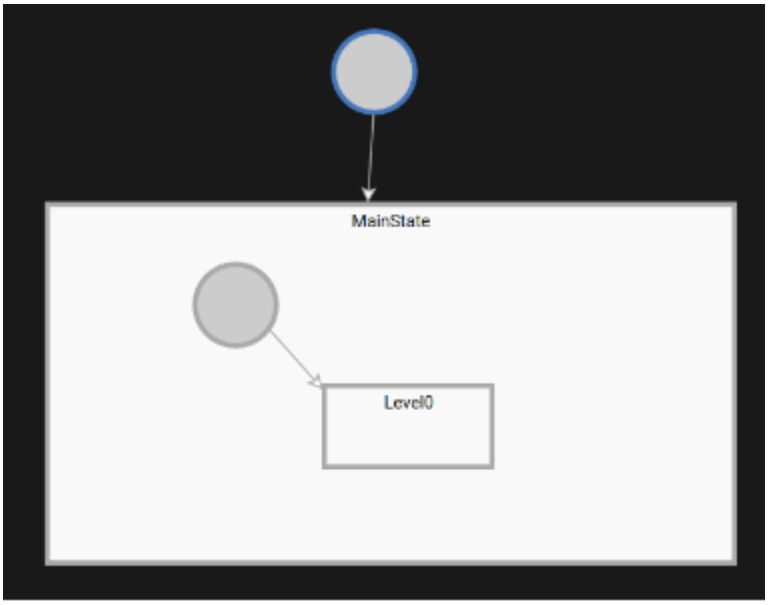
1) Create a new state machine in *Solution explorer* tab as explained in the State Machine Basic Operation documentation.

2) Open the newly created state machine.

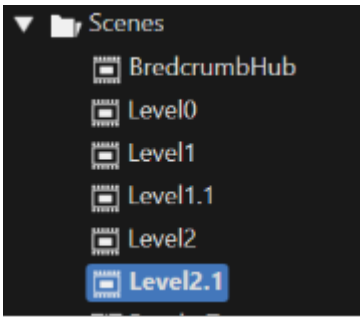
3) Create an Initial State and a State and connect them, it will be the main state that manages other states.



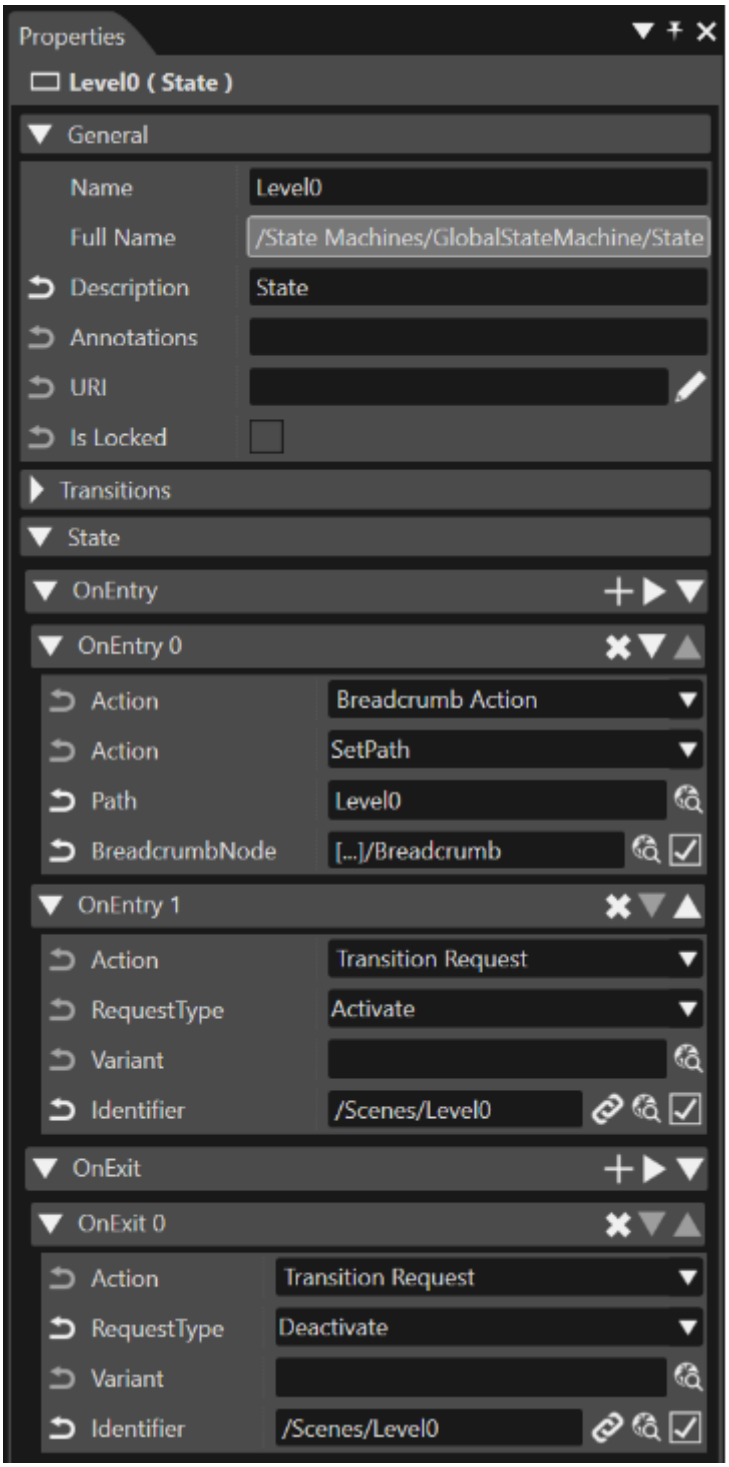
4) Create the Initial State and a State again but this time inside the MainState.



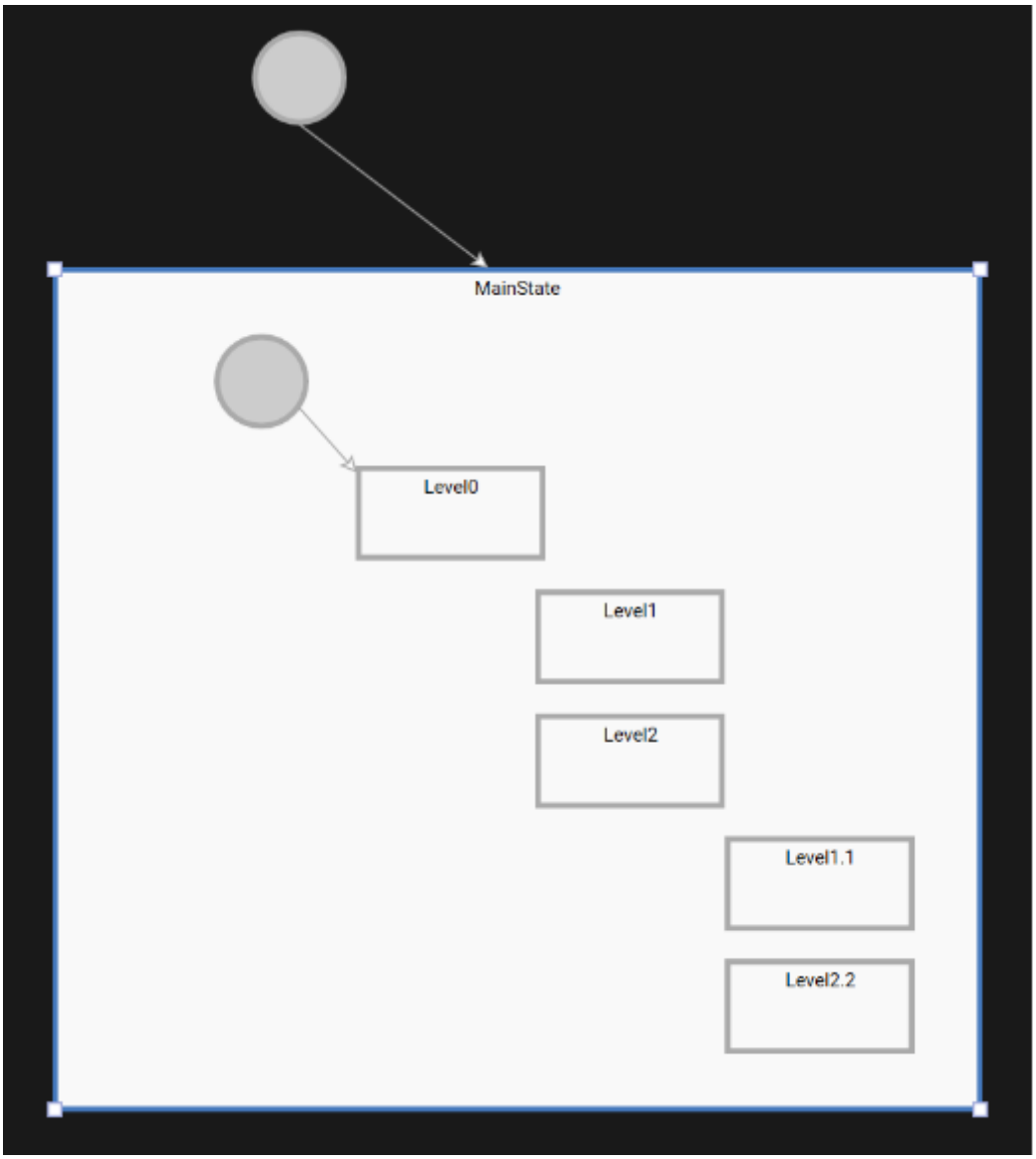
5) Create several sample scenes. On the *BreadcrumbHub* scene add a Breadcrumb control.



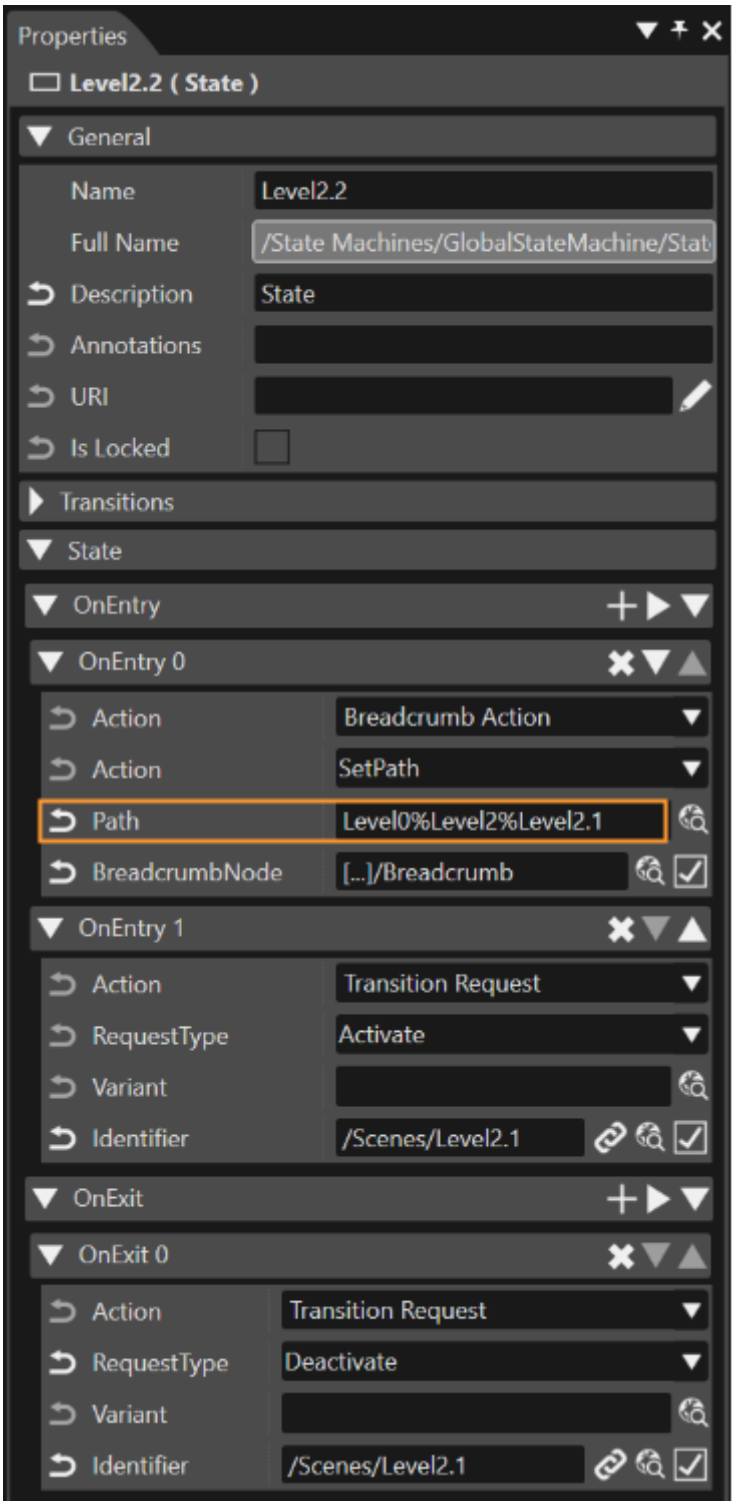
6) In the state machine editor and select the state called "Level0". Set the actions to be called on entry and on exit. At this moment, the path will be set. That scene is first so we only send the name without any separators via Breadcrumb action. Set BreadcrumbNode on BreadcrumbHub scene. Set active / deactivate transition request for selected scene.



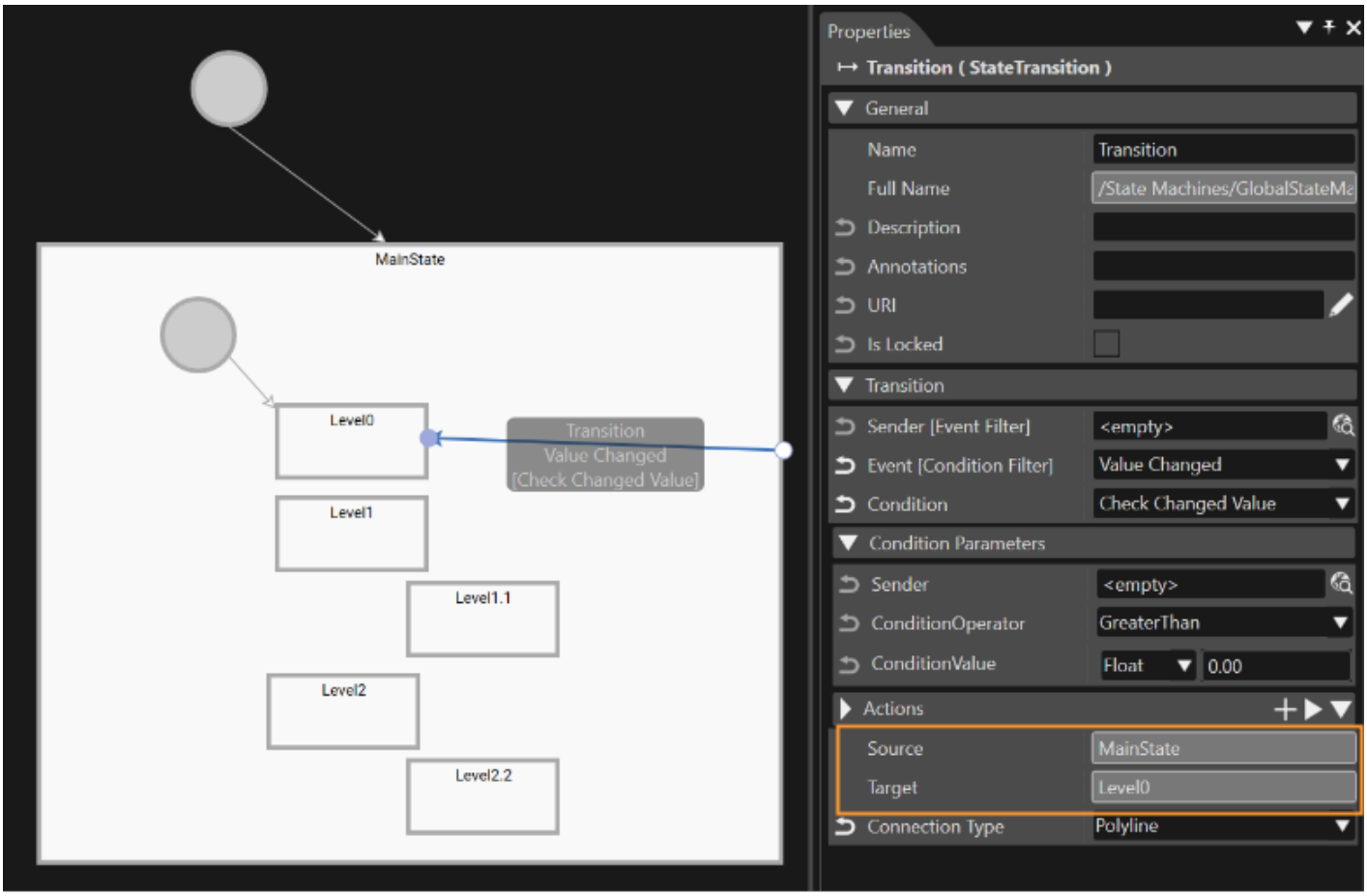
7) Create the next states similarly.



8) Fill each state with actions that are to be called. Below is an example of how to pass nested path with several levels.

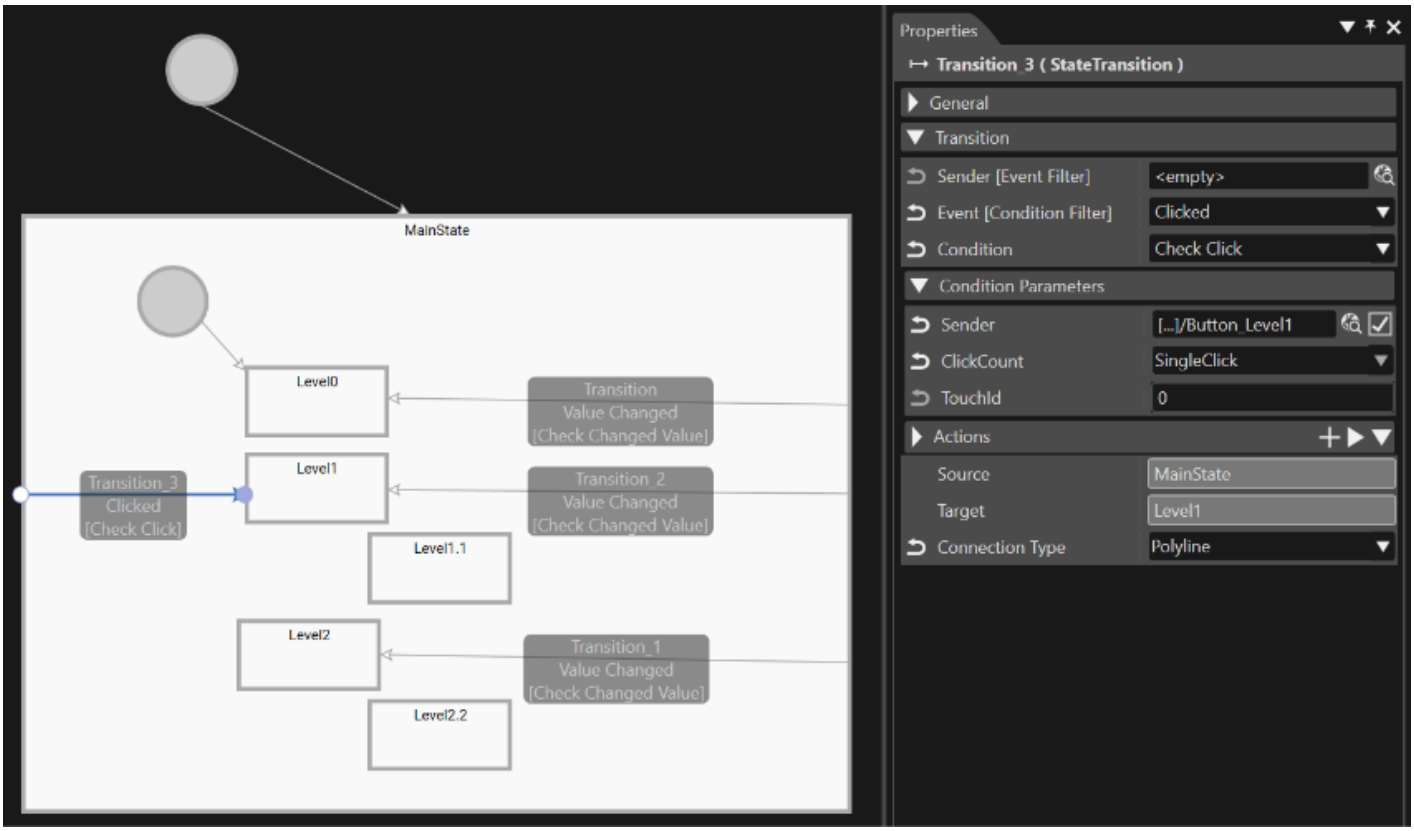


9) After completing all actions for entry and exit in states, we can create transitions when the user clicks on the breadcrumb element. It is a dragging line from the main state to the state we want go back. It should be remembered that we will never go to the final states so Level 1.1 and Level 2.1 will be omitted. On every transition we create a condition. Behavior sends the path from the root to the selected element using ValueChangedEvent.

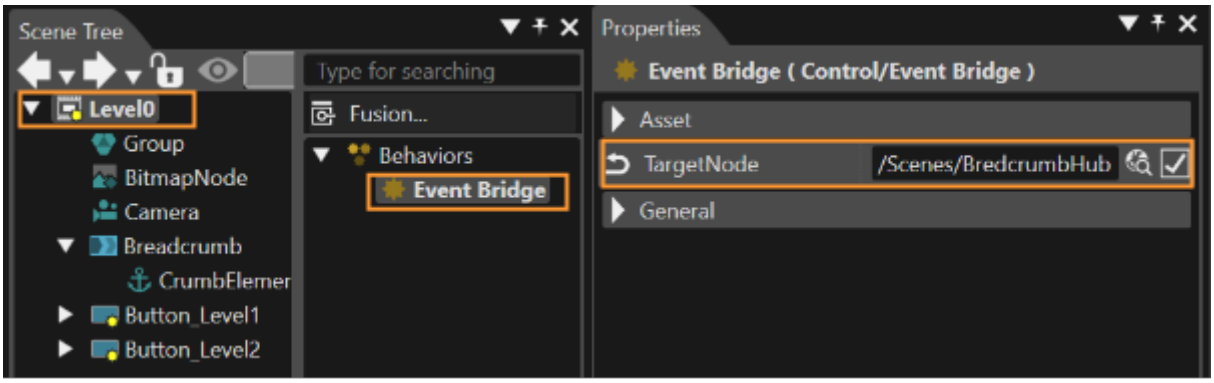


Create similar transition for Level1 and Level2. Don't forget to set the Condition Value to *Level1* and respectively *Level2*.

10) **[optional]** After all transitions with the paths from breadcrumb are defined, you can create transitions that can be used to change the active scene to the next scene by clicking the button on the active scene.



11) The state machine is on the BreadcrumbHub scene so if we want to receive click events in our state machine, we had to add event bridge to other scenes and set target node property to node with state machine.



Sample Solution


<<
Root
Scene1
Scene1.2
Scene1.2.1
Scene1.2.1.2
Scene1.2.1.2
Scenes hierarchy tree
Scene1
Scene1.1
Scene1.1.1
Scene1.1.2
Scene1.1.2.1
Scene1.1.2.2
Scene1.2
Scene1.2.1
Scene1.2.1.1
Scene1.2.1.2
Scene1.2.2
Scene1.2.2.1
Scene1.2.2.1.1
Scene1.2.2.1.1.1