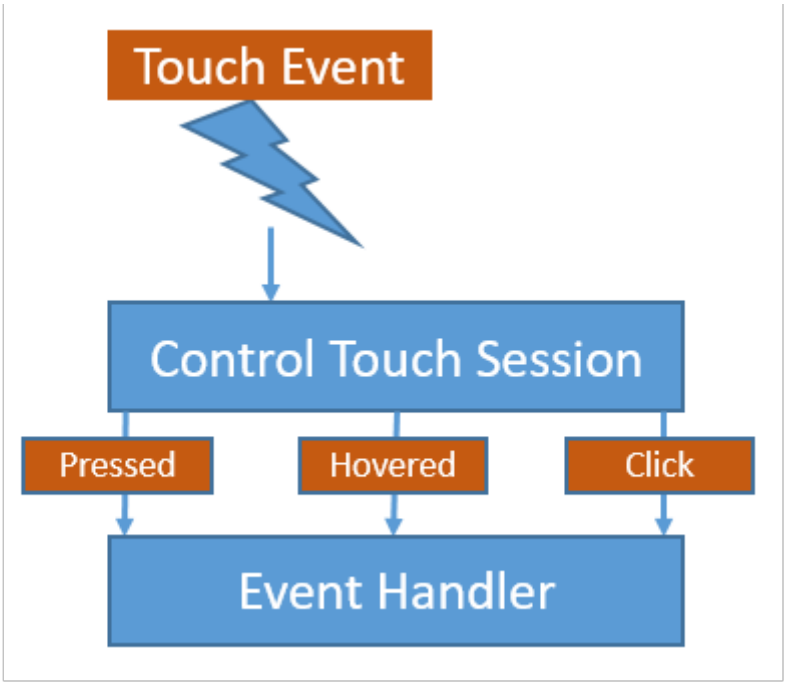
Input Handling

- Touch Input Handling
- Focus Management
- GestureDetector

Touch Input Handling

Description

This chapter gives a detailed, technical overview of how the user touch input is handled internally which is important to understand further concepts like the Control States. The base part therefore is the CgiStudioControl::ControlTouchSession which handles the TouchEvent and emits several Events received by Controls with a Handle Control State Behavior (e.g. Control States).



Touch Event

The Touch Event is emitted by Courier and contains TouchInfo with following states:

- Move
- Down
- Up
- Hover

> Touch input can be tested with the Player which reacts to mouse input and emitsTouch Events.

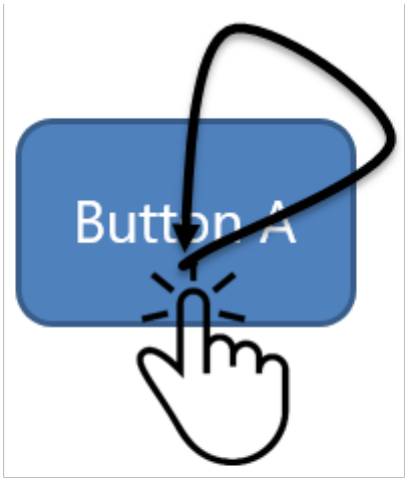
ControlTouchSession

To simplify the process especially for Control developers, the states of a TouchInfo are mapped to several, more specific Events. Therefore the ControlTouchSession generates kind of virtual input events in the sense that a combination of touch events result in a new input event (e.g. Touch Down and Touch Up on the same Control leads to a Click Event). To be able to generate these events and to send them to the right Control the ControlTouchSession contains a list to register all Control State Behaviors. So be sure that each Control which should be handled by the Control Touch Session has to contain a Control State Behavior. For each of the following events it is important to know that the events are generated by the ControlTouchSession but they are dispatched from the Control which gets touched. To understand how Events get sent also check chapter Behaviors and Events .

> Controls must have a Handle Control State Behavior to receive Events from the Control Touch Session even if the Event is not sent locally to this Behavior (e.g. the Click Event).

Pressed Event

Basically the Pressed Event which has two different states is triggered by the users touch down (state: Pressed) or by the users touch up (state: Released).
Furthermore the TouchSession reacts to the move of the touch. So if the touch moves and the primarily touched Control is not touched any longer, the ControlTouchSession emits a Pressed Event with the state Released.
If the touch is still down and enters the primarily touched Control, the TouchSession again emits a Pressed Event with the state Pressed.



A user presses Button A and drags outside and enters the area of Button A again then releases it.
So the Pressed Event will be emitted four times with following states: Pressed, Released, Pressed Released.

In many cases, especially concerning Controls, the Pressed Event gets received by the Handle Control State Behavior (see chapter Control States).

The Handle Control State Behavior even receives the Event via Local Dispatch Strategy (see Behaviors and Events). But the Event also gets dispatched via BubbleStrategy from the Control which enables other Behaviors of the Control to receive the event (e.g. PressedCondition).

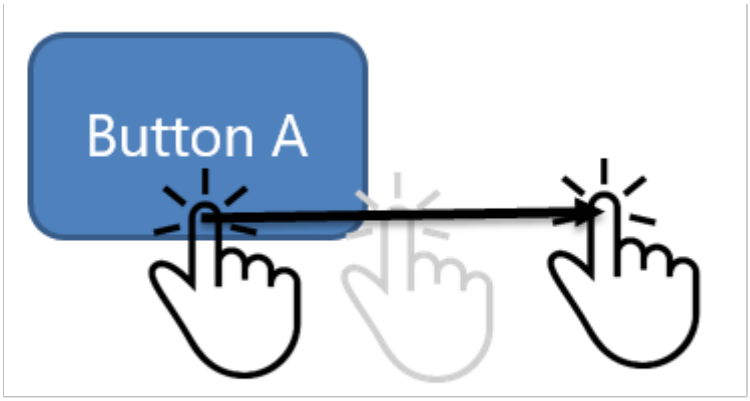
Hovered Event

Emitting the Hovered Event works similar to the Pressed Event. But in difference to the Pressed Event the Hovered Event has an additional state in case of keep hovering over the same object.

Additionally the Event is emitted when the Hover leaves the currently hovered object (**HoverLeave**) and when the Hover enters and object (**HoverEnter**).
As for the Pressed Event also the Hover Event basically gets received by the Handle Control State Behavior (see chapter Control States). But the Event can also be received by another Behavior like the HoveredCondition.

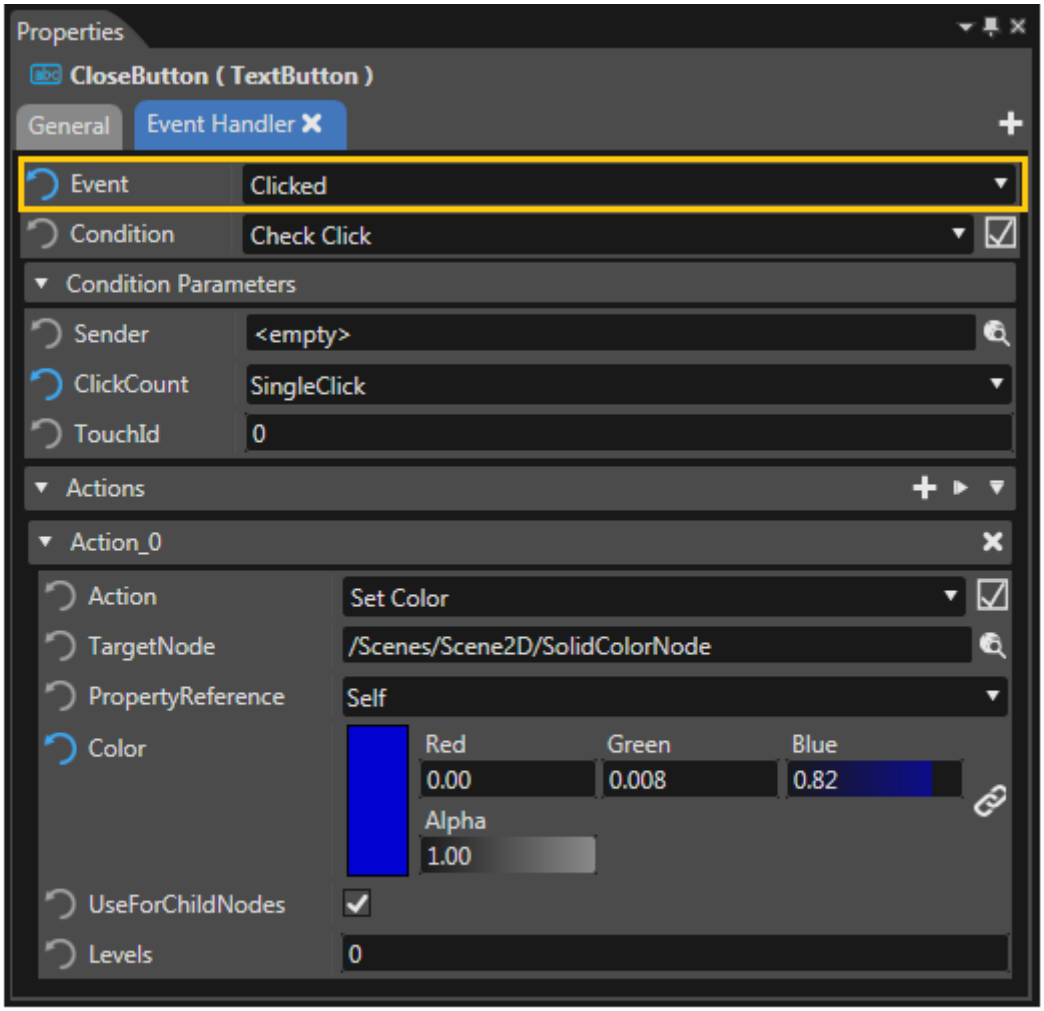
Click Event

As the previously described Events also the Click Event is generated by the ControlTouchSession but it is dispatched from the Control which gets clicked.
But different to the PressedEvent and the HoveredEvent, the ClickEvent is not sent to the Handle Control State Behavior via Local Dispatch Strategy as it is not relevant for the states of a Control. But again the Event gets dispatched from the Handle Control State Behavior which enables the user to directly react on the Event via an EventHandler (also see EventHandler Tab). Furthermore it is important to know that the Click Event is emitted only on release of the touch point. Moreover the the Control of the touch down must be the same as the Control of the touch up. So following example would not trigger a Click Event:



A user presses Button A, drags outside and then releases it.

The Click Event is often used in combination with a EventHandler Tab as shown below:



EventHandler Tab which triggers a Set Color Behavior when receiving a ClickEvent.

> Be sure to read chapter Control States and Control Examples to see how the described Events can be used.

Focus Management

Focus Navigation on Computer

The user of the created HMI wants to move the focus from one control to another. When using the computer, usually a keyboard is used for focus navigation.

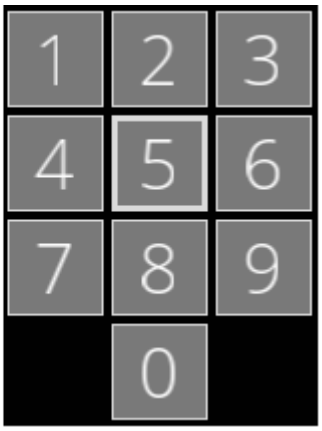
On a computer, the user can use the following keys to move the focus from one control to the other:

- TAB	move the focus to the next control
- CTRL + TAB	move the focus to the previous control

On targets without a keyboard, different input methods can be used for changing the focused control. Automotive targets use the rotary for focus navigation. Therefore, focus navigation is only loosely coupled with a keyboard.

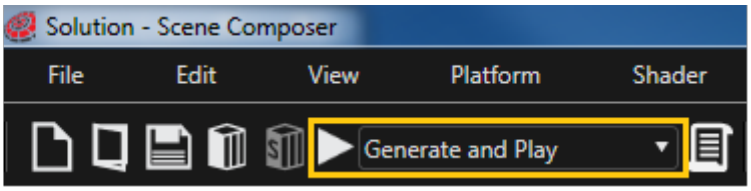
Focus Navigation in CGI Studio

This section uses a phone keypad like the one below to demonstrate the focus changes.



The focused button in the example keypad above is the fifth button (Button 5), whose border is significantly wider.

Focus navigation can be performed in the Player. To start the Player click the play button next to "Generate and Play" in the toolbar below the menubar.



Next/Previous Navigation in CGI Studio

Next Navigation: After selecting a focusable control the focus can be moved to the next focusable control by pressing the **TAB** key of the keyboard. If you repeat pressing the TAB key, the focus will move to the next until the last focusable node is reached. Pressing TAB on the last focusable node will bring the focus back to the first node again.
Focus order when continuing pressing the TAB key: 5-6-7-8-9-0-1-2-3-4-5-6-7-...

Previous: To navigate the focus back to the previous focusable control, press the **CTRL + TAB** keys on the keyboard. If you repeat pressing the CTRL + TAB keys, the focus will move to the previous until the first focusable node is reached. Pressing CTRL + TAB on the first focusable node will bring the focus back to the last node.
Focus order when continuing pressing the CTRL + TAB keys: 5-4-3-2-1-0-9-8-7-6-5-4-3-2-1-...

> If no focusable control has the focus and a key that navigates the focus (e.g. a key of the cursor control pad or the TAB key) is pressed, the focusable control that has the lowest TAB-order gets the focus.

Geometrical Navigation in CGI Studio

Geometrical Navigation means a navigation in one of the following directions: Left, Right, Up, Down, Up Left, Up Right, Down Left, Down Right
On the computer, navigations in those directions can be performed using the cursor control pad or the numeric pad (for diagonal navigation) of the keyboard.

The following describes the focus change that happens when repeatedly clicking the noted navigation key in the example keypad above. Initally, for every noted navigation key, the focus is on the fifth focusable control as in the image of the keypad above.

• **Navigate Left:** 5-4 • **Navigate Right:** 5-6 • **Navigate Up:** 5-2 • **Navigate Down:** 5-8-0		

• **Navigate UpLeft:** 5-1 • **Navigate UpRight:** 5-3 • **Navigate DownLeft:** 5-7 • **Navigate DownRight:** 5-9		
		

Managing the Focusable Nodes

Each keyboard focusable behavior notifies the KeyboardFocusManager about its view getting activated or deactivated.

- *RegisterNavigation* is called when the view gets activated.
- *DeregisterNavigation* is called when the view gets deactivated.

Concept of the Next/Previous Navigation

The logical order of the focusable nodes is defined by the following criteria:

1. The tab order of its scene: This order is stored as a dynamic property and can be set by a *KeyboardFocusTabOrderBehavior*.
2. The asset order of its scene: This order is defined by the order in which the scene is stored in the asset.
3. Its tab order: This order is stored as a dynamic property and can be set by a *KeyboardFocusTabOrderBehavior*.
4. Its tree order: The tree order in pre-order tree traversal.

KeyboardFocusTabOrderBehavior
The tab order of a node is defined as relative or absolute. The default order is *relative*.

- *relative* considering the parent inherited tab order
- *absolute* will stop the inherited tab order

The next/previous navigation updates the logical order for each next/previous navigation event and sorts the focusable nodes, finds the currently focused node in that list and transfers the focus to the next/previous entry in the list.

Concept of the Geometrical Navigation (Left, Right, Up, Down, Up Left, Up Right, Down Left, Down Right)

For each geometrical navigation event the bounding box in the display of the node is calculated per camera in the scene. Each keyboard focusable node has one entry in the lookup list for each camera.

The keyboard focused node is extended by the camera that is responsible for the focus. A scene may have several cameras and the focus will be caused by one of them.

The following events may trigger a change of the keyboard focus:

• Navigate to the Left • Navigate to the Right • Navigate Up • Navigate Down		
		
• Navigate to the UpLeft direction • Navigate to the UpRight direction • Navigate to the DownLeft direction • Navigate to the DownRight direction		
		

	**Navigate to the Left:**

1. Find the nearest node of the nodes that are at least partially within the top and the bottom boundary of the keyboard focused node. The distance of the right node boundary to the left boundary of the keyboard focused node is minimal. Limit the right node boundary to the left boundary of the keyboard focused node. For multiple matches use the same order criteria as for the previous/next navigation and take the first one.
2. If no node was found, find the nearest node (where the corner to corner distance is minimal) of the nodes that intersect the left sectors:

- corner to corner distance for nodes in the top left sector: take the node's bottom right corner (limit that corner to the left boundary of the keyboard focused node) and the keyboard focused node top left corner and calculate the square distance.

- corner to corner distance for nodes in the bottom left sector: take the node's top right corner (limit that corner to the left boundary of the keyboard focused node) and the keyboard focused node bottom left corner and calculate the square distance.
- All other sectors are not on the left of the keyboard focused node.
- For multiple matches use the same order criteria as for the previous/next navigation and take the first one.

	**Navigate to the Right:**

The same as navigation to the left but in this case, left is right, right is left, top is top and bottom is bottom in the description. Imagine a mirrored image.

	**Navigate Up:**

The same as Left navigation but left is bottom, right is top, top is left and bottom is right in the description. This is a rotation by 90° counter clock wise

	**Navigate Down:**

Navigate Down: The same as Left navigation but left is top, right is bottom, top is right and bottom is left in the description. Which is a rotation by 90° clock wise.

	**Navigate to the UpLeft direction:**

Find the nearest node (corner to corner distance is minimal) of the nodes the intersect the top left sector:

- corner to corner distance: take the node's bottom right corner (limit that corner to the left and top boundary of the keyboard focused node) and the keyboard focused node top left corner and calculate the square distance.
- For multiple matches use the same order criteria as for the previous/next navigation and take the first one.

	**Navigate to the UpRight direction:**

The same as UpLeft navigation but left is right, right is left, top is top and bottom is bottom in the description. Imagine a mirrored image.

	**Navigate to the DownLeft direction:**

The same as UpLeft navigation but left is bottom, right is top, top is left and bottom is right in the description. This is a rotation by 90° counter clock wise.

	**Navigate to the DownRight direction:**

The same as UpLeft navigation but left is top, right is bottom, top is right and bottom is left in the description. This is a rotation by 90° clock wise.

GestureDetector

How to implement a custom GestureDetector

The ControlTouchSession supports GestureDetectors. You simply register your own detector and register it at the ControlTouchSession.

- The ClickGestureDetector is one of the existing detectors. It detects click, drag and press related gestures.
- Since it only detects the gestures it is clear that the ControlTouchSession is the only place where it is referenced (and also only because it is registered by default).
- GestureDetector::OnEvent is the only interaction interface with the ControlTouchSession.

The ControlTouchSession will send these events to the gesture detector:

UpdateEvent

This event indicates an update iteration (one per frame)

GestureDetector::TouchInput

This is a single/multi touch event extended by this information:

1. The state of the gesture detector session (FirstDown, LastUp).
2. The behavior that has currently the touch focus (which is the touchable behavior that was hit by the first finger that touched the screen during a gesture detector session)
3. The behavior that is currently touched by the touch event (also updated by Drag).

The following states of the gesture detector session are available:

- FirstDown: The gesture detector session begins when the first finger touches the screen.
- LastUp: The gesture detector session ends when the last finger is removed from the screen.

In between you will get updates in the following cases:

- Down: another finger touched the screen
- Drag: a finger has been moved.
- Up: one finger has been removed from the screen. But at least one more is on the screen.

GestureDetector::AbortEvent

This event tells that another behavior has taken the exclusive touch control. Hence, no gesture detector will participate in the current gesture detector session and therefore has to reset all internal states and wait for the next session.

GestureDetector::DeregisterEvent

The provided behavior is removed from the ControlTouchSession. Hence the gesture detector also has to remove all references to that behavior.

See also:

- Check Click
- Check Drag Drop
- Check Hover
- Check Pressed
- Swipe Gesture Condition
- Transform Gesture
- Transform Gesture Action