Component Events

Event Handlers

When running scripts on a component, we typically don't want it to be constantly running, but instead want the script to trigger when the user does something on screen such as clicks with the mouse. That something the user does is called an *Event* and can range from a simple mouse click, keypress to a window opening, or a component property change. When certain events happen, they trigger Event Handlers, which use a script to *handle* what happens when the event occurs.

This page lists out all of the event handlers that are on Ignition's standard modules. Any third party modules may add new components which may potentially have new event handlers.

On this page

...

Event Object

Every Event Handler contains an **event** object, which allows you to interact with the component and the entire window hierarchy within your script. While each **event** object has different properties depending on what Event Handler it resides in, each **event** object contains a **source** property, which is a reference to the component that fired the event. Using `event.source` not only gives us access to all of the properties available on that particular component, such as the text property of a text field,

Pseudocode - Event Handler Source Component Properties

```
# Here we start with the event object, then use source to go to
the component that fired the event,
# and then use the name of the property to access its value. In
this case, we accessed the text property.
text = event.source.text
```

but it also provides us with a way to navigate to other components within the hierarchy. For example, here we have a script on a button that references a text field.

Pseudocode - Event Handler Other Components

```
# Here again we start with event.source to get to the component
that fired the event, but now we use
# parent to go up to the root container, and then getComponent to
navigate back down to a different component.
text = event.source.parent.getComponent("Text Field")
```

INDUCTIVE UNIVERSITY

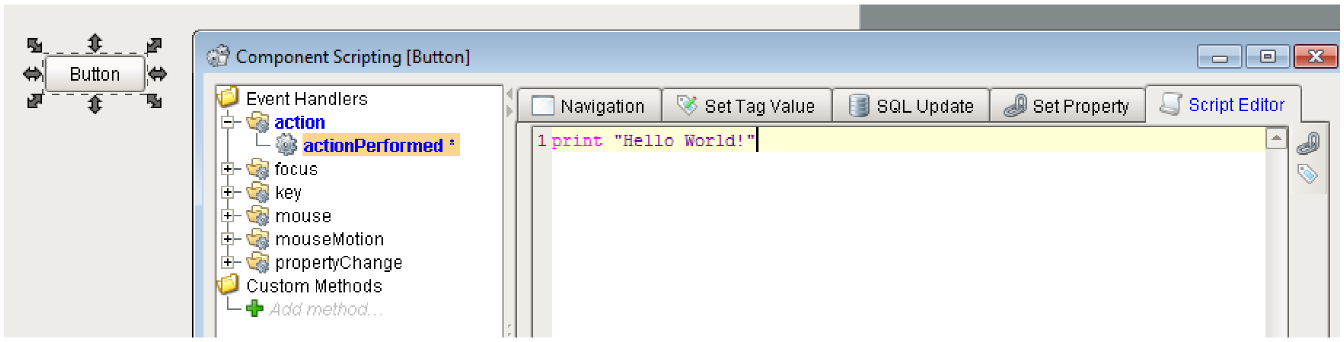
Component Event Handlers

Watch the Video

Even when components are disabled, most scripting events can still occur. For example, a mouse click can still happen on a disabled component, which is why we recommend using the action performed event when placing a script on a button.

Action Event Handlers

The Action category of event handlers pertains to components being "used" from the client, such as a button being pressed or a checkbox component being selected.



```
print "Hello World!"
```

Events

Events	Description
actionPerformed	This event is fired when the 'action' of the component occurs. What this action is depends on the type of the component. Most commonly, this is used with buttons, where the action is that the button was pushed, via a mouse click or a key press. See the component reference for details on what the action means for other components. It is recommended to use this event over mouseClicked whenever possible.

Event Object Properties

Properties	Description
source	The component that fired this event.

Python - Button Action Performed

```
# On the actionPerformed of a button, this will print Hello World! to the console each time the
button is pressed.
print "Hello World!"
```

Property Event Handlers

Property event handlers typically trigger based on the property of a component.

Events

Events	Description
propertyChange	Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Event Object Properties

Properties	Description
source	The component that fired this event.
newValue	The new value that this property changed to.
oldValue	The value that this property was before it changed. Note that not all components include an accurate oldValue in their events.
propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Python - Printing the Changing Property

```
# On the propertyChange of a component, this script will print out the name of the property that is
changing.
print event.propertyName
```

Python - Looking for a Specific Property

```
# It is common to use propertyName to look for specific properties to change. This is a great way
to restrict how often your scripts execute
if event.propertyName == 'text':
  print 'The Text property changed'
```

Mouse Event Handlers

The mouse events all correspond to the clicking and movement of the mouse. They are triggered in the client by an operator interacting with a mouse. Touchscreen monitors will trigger these events when a user touches the screen, but not all touchscreens will fire the mouseEntered and mouseExited events.

Events

Events	Description
mouseClicked	This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
mouseEntered	This event fires when the mouse enters the space over the source component.
mouseExited	This event fires when the mouse leaves the space over the source component.
mousePressed	This event fires when the mouse presses down on the source component.
mouseReleased	This event fires when a mouse button is released, if that mouse button's press happened over this component.

Event Object Properties

Properties	Description
source	The component that fired this event.
button	The code for the button that caused this event to fire. Use the constants `event.BUTTON1`, `event.BUTTON2`, and `event.BUTTON3`.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Python - Printing on a Mouse Enter

```
# On the mouseEntered event of a component, this script will only fire if the mouse enters the
bounds of the component.
print "The mouse is inside the component space!"
```

MouseMotion Event Handlers

The mouseMotion events deal with the motion of the mouse over a component. Not all touchscreen monitors will fire these events.

> mouseMotion events will not trigger when the project is viewed from a mobile project as these gestures are used by the browser/device to zoom or pan.

Events

Events	Description
mouseDragged	Fires when a mouse button is clicked on a component and held, and the pointer is then dragged around.
mouseMoved	Fires when the mouse moves over a component, but no buttons are being held.

Event Object Properties

Properties	Descriptions
source	The component that fired this event.
button	The code for the button that caused this event to fire. Use the constants `event.BUTTON1`, `event.BUTTON2`, and `event.BUTTON3`.
clickCount	The number of mouse clicks associated with this event.
x	The x-coordinate (with respect to the source component) of this mouse event.
y	The y-coordinate (with respect to the source component) of this mouse event.
popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise

Python - Printing on a Mouse Movement

```
# From the mouseMotion event on a component, this will print each time the mouse moves when it is
over the component.
print "The mouse is moving over the component!"
```

Key Event Handlers

The key events all have to do with the user pressing a key on the keyboard.

Events

Events	Description
keyPressed	Fires when a key is pressed and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.
keyReleased	Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.
keyTyped	Fires when a key is pressed and then released when source component has the input focus. Only works for characters that can be printed on the screen.

Event Object Properties

Properties	Description
source	The component that fired this event.
keyCode	The key code for this event. Used with the keyPressed and keyReleased events. Uses the standard Java key codes, see below for more information.
keyChar	The character that was typed. Used with the keyTyped event.
keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN. Uses the standard Java key locations, see below for more information.
altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Python - Printing the Key Released

```
# On the keyReleased event of a component, this will print out the key code of the key that was hit on the keyboard,
# but only on release of the key, and only when the component has focus.
print event.keyCode
```

Java Keys

The key Event Handlers use the Java KeyEvent class, which has unique identifiers for both keys and locations on the keyboard to help differentiate which key is actually being pressed on the keyboard. The numeric codes for each unique location and character can be called from the event object using a constant. For example, the letter "a" has the constant name VK_A. This can then be used to compare against the keyCode value like this:

Python - Checking Specific Key Codes

```
if event.keyCode == event.VK_A:
 print "The key press was a"
```

We listed the locations and some common codes below, but the full list of codes can be accessed by going to https://docs.oracle.com/javase/8/docs/api/java/awt/event/KeyEvent.html.

Some Operating Systems reserve certain keys for certain function, and will capture the key press or release before it gets sent to the Client. For example, many Operating Systems use the TAB key to shift focus to the next field.

Key Code Constants

VK_0 - VK_9	VK_END	VK_PAGE_UP	VK_DOWN	VK_CONTROL

Key Location Constants

KEY_LOCATION_LEFT	KEY_LOCATION_RIGHT	KEY_LOCATION_NUMPAD

VK_A - VK_Z	VK_ENTER	VK_RIGHT	VK_PAGE_DOWN	VK_LEFT
VK_F1 - VK_F24	VK_HOME	VK_SHIFT	VK_UP	VK_TAB
VK_ALT	VK_INSERT	VK_SPACE	VK_ESCAPE	

KEY_LOCATION_STANDARD	KEY_LOCATION_UNKNOWN	

Focus Event Handlers

Focus events deal with focus moving between different components on a window. Opening windows, using tab to move around the screen, or clicking on components will trigger these events. Note that not all components can hold focus.

Events

Events	Description
focusGained	This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.
focusLost	This event occurs when a component that had the input focus lost it to another component.

Event Object Properties

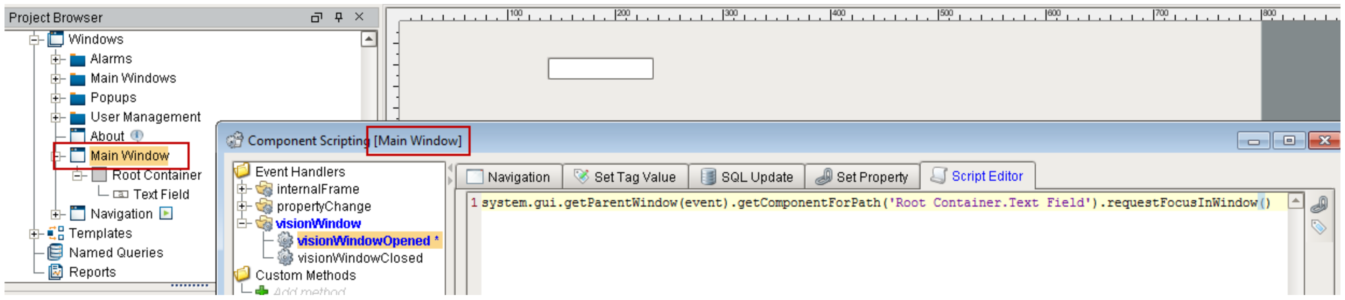
Properties	Description
source	The component that fired this event.
oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

Python - Printing on Focus Gained

```
# On the focusGained event of a few different components, this script can print out when the
component has gained focus.
print "The component name now has focus!"
```

VisionWindow Event Handlers

The visionWindow events are specific to windows and not available elsewhere. Right-Click on the window name in the Project Browser pane and select Scripting to get access to these events. These are triggered by a window opening or closing.



```
system.gui.getParentWindow(event).getComponentForPath('Root Container.Text Field').requestFocusInWindow()
```

Events

Events	Description

visionWindowOpened	This event is fired each time the window is opened and before any bindings are evaluated.
visionWindowClosed	This event is fired each time the window is closed.

Event Object Properties

Properties	Description
source	The vision window that fired this event.

Python - Grabbing Focus on Window Opened in Two Different Ways

```
# From a visionWindowOpened event on a window, you can request the focus of components in the
window, to start the focus on a component other than the upper left most component.

# Here we grab the reference to the component using the property selector on the upper right side
of the script editor.
system.gui.getParentWindow(event).getComponentForPath('Root Container.Text
Field').requestFocusInWindow()

# Here we can manually enter in the path to the component using our knowledge of the component
hierarchy and
# the getRootContainer function. Both of these functions work in the same way.
system.gui.getParentWindow(event).getRootContainer().getComponent("Text
Field").requestFocusInWindow()
```

InternalFrame Event Handlers

The internalFrame events are fired by windows: windows are known as "internal frames" in the underlying Java windowing system that the Vision component uses. Note that the source of these events is the window itself, just like the visionWindow events above.

Events

Events	Descriptions
internalFrameActivated	Fires whenever the window is shown or focused. If you want a script to fire every time a window is opened, use this event.
internalFrameClosed	Fires when a window is closed.
internalFrameClosing	Fires right before a window is closed.
internalFrameDeactivated	Fires when a window loses focus.
internalFrameOpened	Fires the first time a window is opened. Note that when windows are closed and cached, next time they are opened this event will not be fired. Use internalFrameActivated instead.

Event Object Properties

Properties	Description
source	The window that fired this event. Use source.rootContainer to get the root container.

Python - Printing on Frame Activation

```
# From the internalFrameActivated event on a window, this will fire each time the window is
focused, so clicking between two different windows will trigger it.
print "This window is now in focus!"
```

Cell Event Handlers

The cell event is unique in that it only appears on the Table component. It will trigger when something within a cell changes, and once for each cell changed.

Events

Events	Description
cellEdited	This event is fired when one of the cells in a table component has been modified.

Event Object Properties

Properties	Description
source	The table component that fired this event.
oldValue	The old value in the cell that changed.
newValue	The new value in the cell that changed.
row	The row of the dataset this cell represents.
column	The column of the dataset this cell represents.

Pseudocode - Updating a Database Table

```
# From the cellEdited event of a table component, this script can update our database table with
any new data that is entered in the table

# Get the id of the row we edited and the headers
id = event.source.data.getValueAt(event.row, 'id')
headers = system.dataset.getColumnHeaders(event.source.data)

# Build our Update query.
query = "UPDATE User SET %s = ? WHERE id = ?" % (headers[event.column])
args = [event.newValue, id]

# Run the query with the specified arguments.
system.db.runPrepUpdate(query, args)
```

Item Event Handlers

The item event is unique in that it only appears on components that can be "on" or "off, such as with Radio Buttons, Check Boxes, and Toggl e Buttons.

Events

Events	Description
itemStateChanged	This event fires when the state of the component changed. This event is the best way to respond to the state of that component changing.

Event Object Properties

Properties	Description
source	The component that fired this event.

stateChange	An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.
SELECTED	The constant that the stateChange property will be equal to if this event represents a selection.
DESELECTED	The constant that the stateChange property will be equal to if this event represents a de-selection.

Python - Printing on a Radio Button Selection

```
# On the itemStateChanged event of a radio button, this will print when this specific radio button
is selected.
if event.stateChange == event.SELECTED:
 print "This radio button is selected!"
```

Paint Event Handlers

The paint event is only found on the Paintable Canvas component, and is used to customize how the component gets painted. This event requires a heavy knowledge of programming using the Java 2D drawing tools, but there is code for a pump shape each time you add a new Paintable Canvas to a window.

Events

Events	Description
repaint	This event will fire whenever the component needs to repaint itself. It will repaint when any of its custom properties change, or when `.repaint()` is called on it. When a Paintable Canvas is first dragged onto the screen, the repaint Event Handler will be filled with an example that draws out a pump.

Event Object Properties

Properties	Description
source	The Paintable Canvas component that fired this event.
graphics	An instance of java.awt.Graphics2D that can be used to paint this component. The point (0,0) is located at the upper left of the component.
width	The width of the paintable area of the component. This takes into account the component's border.
height	The height of the paintable area of the component. This takes into account the component's border.

Python - Painting a Circle

```
# On the repaint event of a paintable canvas component, this will create a circle with a gradient
background color of orange and white.

from java.awt import Color
from java.awt import GradientPaint
from java.awt.geom import Ellipse2D

g = event.graphics

# Body
innerBody = Ellipse2D.Float(8,8,72,72)

#### Scale graphics to actual component size
dX = (event.width-1)/100.0
dY = (event.height-1)/100.0
g.scale(dX,dY)

# Paint body
g.setPaint(GradientPaint(0,40,Color.WHITE, 0,100, Color.ORANGE,1))
g.fill(innerBody)
g.setColor(Color.ORANGE)
g.draw(innerBody)
```

Related Topics ...

- Script Builders