Handling a Datapoint Event

You can receive notification of updates to an input datapoint, and you can receive notification of successful or unsuccessful completion of output datapoint updates that you initiated.

To read the most recent value received with an input datapoint, read the datapoint's data attribute. To receive notification of a new input value for an input datapoint, subscribe to the onUpdate() event for the datapoint.

To receive notification of successful or unsuccessful completion of a previously initiated propagation of an output datapoint, subscribe to the **onComplete()** event.

This section consists of the following:

Device Datapoints Events

To receive notification of a successful completion of a device datapoint update, create a datapoint update event handler function, and register this event handler with the datapoint. To register an update event handler with a device input datapoint, use the **onUpdate()** attribute. To register a completion event handler with a device output datapoint, use the o**nComplete()** attribute.

Example 1

```
#include "IzotDev.h"

SNVT_volt(input) input; //@Izot datapoint(direction=Input) onUpdate(updateEvent)
SNVT_volt(output) output ; //@Izot datapoint onComplete(completionEvent)

void updateEvent(const unsigned index, const IzotReceiveAddress *constpSource) {
// Example number crunching algorithm:
        output.data = 3 * input.data;
        IzotPropagate(output);
}

void completionEvent (const unsignedindex, const IzotBoolsuccess) {
        // The 'output' datapoint completed  propagating.
        // For example, maintain success/failure  rates and raise an error
        // when the error rate exceeds a reasonable  percentage  of the total
        // number of updates.

        ...
}
```

You do not have to define the prototype for your event handler. The handler prototypes are automatically generated within **IzotDev.h**.

Your handler implementations must match the event handler types expected by the IzoT Device Stack, as detailed in **IzotTypes.h**.

The **updateEvent** and **completionEvent** handlers, shown in the example above, are supplied with an index argument, which can be used to identify the corresponding datapoint by inspecting the datapoint's global index attribute. This allows using one update event handler with multiple datapoints. For event handlers registered only with one datapoint (as shown in the example) the value of the index argument always matches the datapoint's global index value.

Block Datapoint Events

To register an onUpdate or onComplete event for a member of a block, use the onUpdate() and onComplete() attributes with two arguments: the first argument identifies the block member just as with the implement() attribute, the second identifies the event handler in the same way as the device datapoint events discussed above.

Example 2

This example defines two blocks, an implementation of a standard open loop sensor and an open loop actuator block, using the **SNVT_temp_fstandard** datapoint type for both. The application then defines the update event handler with the **actuator_update()** function (whose prototype must match a function prototype defined by the IzoT Device Stack API), and registers this update event handler. When new data arrives for the actuator's **nviValue** member, the event handler executes, then multiplies the new input value with the current value of the **nciGain** property, and writes that result to the sensor's **nvoValue** output. The event handler notifies the IzoT Device Stack that this output datapoint has been updated and is ready for propagation.

```
#include "IzotDev.h"

SFPTnodeObject (node) nodeObject;    //@Izot block
SFPTopenLoopSensor(sensor,SNVT_temp_f) sensor;    //@Izot block implement(nciGain) SFPTopenLoopActuator
(actuator,SNVT_temp_f)actuator; //@Izot block onUpdate(nviValue,actuator_update)

void actuator_update(const unsigned index, const IzotReceiveAddress *constpSource) {
        SNVT_amp_f  value =actuator - .nviValue - .data;
        value *=sensor.nciGain->multiplier;
        value /=sensor. nciGain->divisor;
        sensor.nvoValue.data = value;
        IzotPropagate(sensor.nvoValue);
}

int main(void) {
        IzotInit();
        while (1) {
                IzotEventPump();
        }
        return 0;
}
```

The application accesses block datapoint members through the member name as defined within the profile, e.g., **sensor.nvoValue**, but accesses block property members through property pointers, such as **sensor.nciGain**.

Additional Considerations

You can share event handlers and re-use them within the same event type, but you cannot share one event handler among different event types, even if the function prototype requirements are met for all event types.

You can explicitly re-use an event handler by referencing it in more than one **onUpdate()** or **onComplete()** event registration.

Example 3

This example explicitly shares the **update_occurred()** handler between the **volt[0].nviValue**, **volt[1].nviValue**, and **amp.nviValue** datapoints.

```
#include "IzotDev.h"

SFPTnodeObject(node)  nodeObject;       //@IzoT block
SFPTopenLoopActuator(volt, SNVT_volt) volt[2]; //@IzoT block onUpdate(nviValue, update_occurred)
SFPTopenLoopActuator(amp, SNVT_amp) amp;          //@IzoT block onUpdate(nviValue, update_occurred)
```

You can re-use an event handler by applying it to a declaration of more than one item.

Example 4

This example implicitly shares the **update_occurred()** handler between the **volt[0].nviValue** and **volt[1].nviValue** datapoints.

```
#include "IzotDev.h"

SFPTopenLoopActuator(volt, SNVT_volt) volt[2]; //@IzoT block onUpdate(nviValue, update_occurred)
```

When an event handler applies to exactly one item. You do not have to sample the index argument provided to the event handler. A simple event dispatcher is automatically generated, which executes your event handler only for those datapoints to which you applied the handler.