Handling a Device Event

You can receive notification of events that apply to the device. The device events are the following:

- Offline and Online events
- Reset event
- Wink event
- Service event

You can register none, one, or multiple event handlers to each of these events.

This section consists of the following:

Offline and Online Events

You can be notified of requests to change the device application from the offline state to the online state, or from the online state to the offline state. The online state is used for normal device operation and communication. The offline state is typically requested by a network management tool. It is used to suspend application network communication and is typically used for doing device or network maintenance, or to disable a device that as failed but continues to communicate. The application may continue to run in the offline state, but should receive but ignore input datapoint updates and not generate output datapoint updates. While offline, applications must continue to maintain any physical I/O in a safe state. A network management tool will also typically be used to change a device from the offline state back to the online state.

To respond to a request to change from the online state to the offline state, define an offline event handler for the **onOffline()** event and register your handler using the ***//@IzoT* Event** directive. You do not have to specify your handler's function prototype, but you must implement the handler as a function with a prototype matching the type defined by the IzoT Device Stack API, and you must implement the handler as a **global** (non-static) function.

Offline and Online Event Example

```
#include "IzoTDev.h"

//@IzoT Event onOffline(my_offline_handler)

void my_offline_handler (void) {
        // Handle the transition into the offline state
}
```

The **onOnline()** event occurs when the device is taken online. To respond to a request to change from the offline state to the online state, define an online event handler for the **onOnline()** event and register your handler using the ***//@IzoT* Event** directive. You do not have to specify your handler's function prototype, but you must not implement the handler as a static function.

The **onReset()** event implies **onOnline()**.

Reset Event

You can be notified when the IzoT Device Stack resets. This is useful for executing any application code required on application start-up, or in response from a network management tool to reset the device. A network management tool may reset a device to re-configure or reset application algorithms or state machines. With the IzoT Device Stack DX, this event does not require a restart of your application or a physical reset of your device hardware.

To implement a reset event handler that is executed upon a request for an application reset, define a reset event handler for the **onReset()** event and register your handler using the ***//@IzoT* Event** directive. While you do not have to specify your handler's function prototype, you must implement the handler with a prototype matching the type defined by the IzoT Device Stack API, and you must implement the handler as a **global** (non-static) function.

Reset Event Example

```
#include "IzotDev.h"

//@Izot Event onReset(my_reset_handler)

void my_reset_handler(void) {
        // Handle the reset
}
```

Wink Event

You can make your devices easier to discover at installation time by adding support for winking the device. A wink is a visible or audible indication from a device that an installer can request via a network management tool. This is useful when more than one of your devices are discovered on a network, and the network integrator needs to determine the mapping from physical devices to the logical devices defined in or discovered by the network management tool. To support winking, add an event handler for the onWink() event to your application. Within your event handler, implement code to safely provide a visual or audible cue to a network integrator upon requesting a wink via a network management tool. For example, you can flash an LED or activate a buzzer for a short period of time in response to a wink request. You can optionally provide a method for a user to cancel winking, for example by pressing a button on the device while it is winking.

To implement a wink event handler that is executed up receiving a wink request, define a wink event handler function and register your handler using the **//@IzoT Event** directive. You do not have to specify your handler's function prototype, but you must implement the handler as a function with a prototype matching the type defined by the IzoT Device Stack API, and you must implement the handler as a **global** (non-static) function.

Wink Event Example

```
#include "IzotDev.h"

//@Izot Event onWink(my_wink_handler)

void my_wink_handler(void) {
        // wink, wink, wink
}
```

Service Event

You can be notified when the state of your Service indicator must change. The Service indicator indicates the global state of a device. It is often implemented as an LED, typically yellow, but implementations may also emulate the functionality using a different user interface. For example, an animation shown on a graphical display can implement the same functionality.

The service event supplies information for both implementations using a simple physical output to drive an LED and implementations implementing a logical indicator using a different physical interface. The IzoT Device Stack fires the optional service event whenever the state of a Service LED needs to change (to support applications driving physical LEDs) and when the device stack's state changes (to support applications implementing logical indicators).

To implement a service event, define a service event handler for the **onService()** event and register your handler using the **//@IzoT Event** directive. While you do not have to specify your handler's function prototype, you must implement the handler as a function with a prototype matching the type defined by the IzoT Device Stack API, and you must implement the handler as a **global** (not-static) function.

Service Event Example 1

```
#include "IzotDev.h"

//@IzoT Event onService(my_service_led_handler)

void m_service_led_handler(void) {
        // Drive the logical LED based on the state value or the physical one
        // based on the physical state argument.
}
```

Additional Considerations

You cannot reuse an event handler for multiple event types. You can register multiple event handlers for the same event. For example, you can register two offline event handlers as shown in the following example.

Service Event Example 2

```
#include "IzotDev.h"

//@IzoT Event onOffline(offline_1), onOffline(offline_2)

void offline_1(void) {
        // Handle the transition into the offline state
}

void offline_2(void) {
        // Handle the transition into the offline state
}
```

Multiple device event handlers are executed in the order in which they are declared. In Example 2 above, **offline_1()** executes before **offline_2()**.