Implementing a Device Datapoint

You can implement a device datapoint to provide a network input to your application or a network output from your application. Device datapoints are datapoints that are not members of a block. Because blocks provide grouping of datapoints and properties into logical entities with well-defined behavior, device datapoints are not commonly used. However, there are cases when device datapoints can be useful, and they are often used for rapid prototyping or testing. See Implementing a Block for details about implementing blocks and datapoints or properties within blocks.

To implement a device datapoint, use the IML datapoint class. You can use the direction modifier to specify the direction of the datapoint. It accepts Input and Output (the default). You can specify other attributes to support the provisioning of a non-zero initial value and advanced controls over the use of protocol features such as authentication, priority, service type, or data persistence.

Example 1

This example implements a pair of two device datapoints using the **SNVT_temp_f** datapoint type (an IEEE 754 single precision floating point variable used for temperature values in degrees Celsius). The **nvoTemperature** datapoint uses the default data point direction, which is output. The **nviTemperature** datapoint is implemented with the optional **direction=Input** modifier, which defines it as an input datapoint.

```
#include "IzoTDev.h"

SNVT_temp_f(nvoTemperature)nvoTemperature ; //@Izot datapoint
SNVT_temp_f(nviTemperature)nviTemperature ; //@Izot datapoint(direction=Input)
```

When you build this application, the IzoT Interface Interpreter generates the following C type definitions within the **IzotDev.h** file for this datapoint declaration:

```
typedef float SNVT_temp_f;
#define SNVT_temp_f (m) SNVT_temp_f_ ##m
typedef struct {
        SNVT_temp_f data;
        unsigned short global_index;
} SNVT_temp_f_nvoTemperature ;
extern SNVT_temp_f_nvoTemperature nvoTemperature;
#define SNVT_temp_f(m) SNVT_temp_f_ ##m
typedef struct {
        SNVT_temp_f data;
        unsigned short global_index;
} SNVT_temp_f_nviTemperature;
extern SNVT_temp_f_nviTemperature nviTemperature;
```

You must call the **IzotInit()** function before you access the IzoT Device Stack or elements of your application's interface. This function automatically clears the device datapoint's data value and assigns the correct values to the datapoint's **global_index** property. The **global_index** property is a read-only property.

Example 2

The following IML example declares a voltage output datapoint based on the **SNVT_volt_f** datapoint type. This type is an IEEE 754 single-precision floating point variable to exchange values in units of electrical Volts.

```
#include "IzotDev.h"

SNVT_volt_f(volts) volts; //@IzoT Datapoint
```

The IzoT Interface Interpreter generates the following definitions for this datapoint declaration:

```
typedef float SNVT_volt_f;
#define SNVT_volt_f(m) SNVT_volt_f_ ##m
typedef struct {
        SNVT_volt_f data;
        unsigned short global_index;
}SNVT_volt_f_volts;
extern SNVT_volt_f_volts volts;
```

You can access the datapoint's current value with the data field, or read the additional attributes for further details. Those additional attributes are initialized within the **IzotInit()** function and are read-only.

The name of the global C variable, **nviTemperature** and **nvoTemperature** mentioned in Example 1, appears twice in each declaration, once for the name of the global variable, and once as a modifier to the datapoint type. The general syntax of the left side of a device datapoint declaration is this:

```
type-name(modifier) variable(s); //@Izot datapoint...
```

The **type** modifier is required to generate a unique name for the device datapoint's type. It is not required to equal the name of the variable so long as the modifier is unique within this application and this datapoint type, but you can use the variable's name to simplify naming.

See IML Syntax Summary for a complete list of modifiers and attributes supported with the device datapoint implementations.

See Data Types for more about the IML data types.