Implementing a Block

You can implement a block to provide a network visible component of your application. The block encapsulates the datapoints and properties required for a task performed by your device application. Each block is defined by a *profile* that defines the datapoint and property members that can be implemented by the block. A profile defines mandatory and optional members. A block always implements the mandatory members, and may also implement any of the optional members.

To implement a block, use the IML block class. The code generated from your IML contains all the mandatory datapoint and property members defined in the profile implemented by the block. You can use the **implement()** attribute to add optional profile members into your implementation of the block, or to refine mandatory members with application-specific details, such as application-specific initial values.

Multiple implementations of the same profile may require different block type definitions. For example, you can implement an **Open Loop Sensor** block with a **SNVT_volt_f** datapoint type, and implement another **Open Loop Sensor** block using a **SNVT_amp_f** datapoint type. The IzoT Interface Interpreter creates the named datatype for the block from the profile name, the type name modifier which you supplied in the IML declaration, and the name of the datapoint type used to resolve **SNVT xxx** references within the profile. You can use the block's name for the type name modifier to ensure that multiple blocks based on the same profile have unique named datatypes. To implement multiple blocks from the same profile, but with different implementation details, use a different declaration as shown in Example 5.

Devices which implement more than one block must also implement the standard **Node Object** profile. The **Node Object** block acts as the housekeeper for simple tasks common to all blocks on the device, such as enabling or disabling them. You must implement the **Node Object** block's functionality as detailed in the **Node Object** profile. A default **Node Object** implementation is automatically generated when you create a new project with the IzoT CPM 4200 SDK.

To implement a **Node Object** block explicitly, or to promote optional profile members, declare your **Node Object** using the same block declaration syntax discussed above.

You must declare your **Node Object** block before any other block declarations.

Some profiles define one or more of their datapoint members using the generic **SNVT_xxx** placeholder type. To specify the datapoint type for a block that uses the **SNVT_xxx** type, specify the datapoint type as the second parameter when you create the block. All **SNVT_xxx** references within a given profile are implemented using the same actual datapoint type. To implement a profile with multiple **SNVT_xxx** members such that each **SNVT_xxx-typed** member implements a different datapoint type, derive a user-defined profile and override the affected members with the desired specific datapoint types.

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

See Data Types for more about the IML data types.

Example 1

This example implements a **Node Object** and two profiles as arrays of three block search. The first implements the **SFPTopenLoopActuator** profile. The second implements the **SFPTopenLoopSensor** profile and adds the optional **nciGain** property with an initial value of 12, 13 to the implementation. Both profiles define a generic datapoint type for the principal block datapoint member. The **SNVT_xxx** placeholder type is specified as a **SNVT_amp_f** type in the example.

```
#include "IzoTDev.h"

SFPTnodeObject(node) nodeObject;       //@izot block
SFPTopenLoopActuator(actuator, SNVT_amp_f) actuator[3];    //@izot block
SFPTopenLoopSensor(sensor, SNVT_amp_f) sensor[3];   //@izot block implement(nciGain, init={12, 13})
```

This example declares arrays of three block-typed variables (sensor and actuator), implemented three actuator and three sensor blocks. Blocks implemented within the same declaration share common attributes. For example, all three sensor blocks implement an **nciGain** property with an initial value of 12, 13, and all actuator and sensor blocks implement the generic datapoint member using a **SNVT_amp_f** datapoint type.

Example 2

This example implements a **boiler** block based on the **SFPTboilerController** profile.

```
#include "IzotDev.h"

SFPTboilerController(boiler)boiler;    //@IzoT block
```

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

```
#define SFPTboilerController (mod)   SFPTboilerController_  ##mod

typedef struct {
        unsigned short global_index;

        struct {
                SNVT_switch data;
                unsigned short global_index;
        } nvoBoilerState;

struct {
        SNVT_switch data;
        unsigned short global_index;
} nviBoilerEnable;

struct {
        SNVT_temp_p data;
        unsigned shortglobal_index;
} nvoEffectSetpt;

SCPTheatSetpt *nciHeatSetpt ;
```

Datapoint members such as **nvoBoilerState** are implemented using the same type as discussed under Implementing a Device Datapoint. Property members such as **nciHeatSetpt** are implemented using property pointers as described in Implementing a Device Property.

Example 3

This example declares a sensor block based on the **SFPTopenLoopSensor** profile. The datapoint type for the data datapoint is declared as a **SNVT_temp _f** floating point temperature value. The optional **nciGain** property member is added.

```
#include "IzotDev.h"

SFPTopenLoopSensor(sensor ,SNVT_temp_f)sensor ; //@Izot block implement(nciGain)
```

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

```
#define SFPTopenLoopSensor(mod, xxx)   SFPTopenLoopSensor_  ##mod_##xxx

typedef struct {
        unsigned short global_index;
        struct {
                SNVT_temp_f  data;
                unsigned short global_index ;
        } nvoValue;
        SCPTgain *nciGain ;
} SFPTopenLoopSensor_sensor_SNVT_temp_f;

extern SFPTopenLoopSensor_sensor_SNVT_temp_f sensor;
```

Example 4

This example declares a function of a pointer to the block from the previous example.

```
#include "IzotDev.h"

SFPTopenLoopSensor(sensor,SNVT_temp_f) sensor;    //@Izot block implementation(nciGain)

void sense (SFPTopenLoopSensor(sensor, SNVT_temp_f) *pBlock, signed long current)
{
        SNVT_temp_f value = current;
        current *= pBlock->nciGain->multiplier / pBlock->nciGain->divisor;
        pBlock->nvoValue.data = current;
}
```

Example 5

This example implements two standard open loop actuator profiles with different implementation details—one uses the **SNVT_volt_f** datapoint type and the other uses the **SNVT_amp_f** datapoint type for the generically-typed profile member.

```
#include "IzotDev.h"

SFPTnodeObject(node) nodeObject; //@Izot block external("Node")
SFPTopenLoopSensor(sensor, SNVT_volt_f) voltSensor;    //@IzoT block external("Volt")
SFPTopenLoopSensor(sensor, SNVT_amp_f) ampSensor;  //@IzoT block external("Ampere")
```