IML Syntax Summary

The IzoT Markup Language (IML) consists of C comments that start with a **//@IzoT** tag. A **//@IzoT** string embedded within a C string constant or another C comment is not treated as an IML tag. The **//@IzoT** tag is not case sensitive, though many IML elements are case sensitive.

An IML declaration consists of the code to the left of the **//@IzoT** tag, the class name which follows, and all further attributes supplied in the remainder of the statement. You may use standard C comments within an IML statement.

The source to the left of the **//@IzoT** tag is required by most IML declarations and generally declares one or more global C variables based on an IML-compatible type.

All variables implemented within the same declaration share the same attributes and modifiers, which are provided to the right of the **//@IzoT** tag.

The general form of an IML declaration is as follows:

```
type(modifier(s)) variable(s); //@IzoT class(modifier(s)) attribute(argument)...
```

The IML class names are block, datapoint, event, property, tag, and option. Class names are case insensitive. Each class has different requirements on type and class modifiers, and supports a different set of attributes.

You can split a single declaration over multiple lines using the standard C "\" line continuation character as the last character on the previous line. You must repeat the **//@IzoT** tag at the beginning of every continuation line. Failure to do so causes the C compiler to incorrectly read the declaration's attributes and generate errors or incorrect code.

Here is an example of a block implementation spanning multiple input lines. The example includes line numbers, starting at number 17.

```
17:  SFPTopenLoopSensor(sensor ,SNVT_volt_f)sensor[8]; //@IzoT Block \
18:      //@IzoT implement(nciGain, init={12, 13}), implement(nciLocation) \
19:      //@IzoT external("sensor")
```

The IzoT Interface Interpreter reports IML syntax errors and warnings for any line of a multi-line IML declaration with the line in which the IML declaration was started. For example, this means that an error encountered when processing the external attribute supplied on the original line number 19 in the above example will be reported as an error occurring in line 17.

Multi-line IML declarations may trigger a C compiler warning about multi-line comments. You can ignore this warning, suppress it, or merge the multiline instruction into one line. If you use the GNU C/C++ compiler included with the CPM SDK, you can suppress this warning with the **-Wno-comment** compiler option.

This section consists of the following:

Block

The **//@IzoT Block** directive creates one or more blocks from a profile. The Block instruction syntax is as follows:

```
block-declaration  ::= profile (profile-modifier) variables ; //@IzoT Block[(locator) [block-attributes ]
```

where

```
profile-modifier ::= type-modifier | type-modifier, snvt-xxx-type
type-modifier ::= name
xxx-type ::= datapoint type
variables ::= variable | variable, variables
variable ::= name | name[number]
locator ::= location="Izot Python Resources package path"
block-attributes ::= block-attribute | block-attribute, block-attributes
block-attribute ::= external(external-name)
        | implement(member-name [, array=array-size]  [, init=initial-value])
        | onComplete(member-name, event-handler)
        | onUpdate(member-name, event-handler)
```

Block Examples

```
SFPTboilerController(boiler)boiler; //@IzoT Block
UFPTmyProfile(example)exampleA,exampleB[4]; //@IzoT Block(location="myResources.resources")
SFPTopenLoopSensor(sensor, SNVT_volt_f)sensor ; //@IzoT Block external("sensor")
```

Block Syntax Elements

Syntax Element	Purpose
profile	The name of the profile, e.g., **SFPTopenLoopSensor**,**UFPTheatExchanger**.
type-modifier	A name or number which, in combination with the profile name (and, if provided, the **xxx-type** name, produces a unique name for the block type. You can use the block variable name for the type modifier. When a declaration implements multiple variables, you can use the name of the first.
xxx-type	The datapoint type to use for all generically typed (**SNVT_xxx**) profile members. This type is only required if the profile implements generic datapoint members.
variable	Name of the global C variable which implements this item. The optional external name for this item. This name appears in the device's public network interface and helps the network integrator to identify the functionality you provide.
external-name	When a block's external name is provided, all members are automatically implemented with external names derived from the member names defined in the profile. When a block's external name is not provided, block members have no external name. Assigning an external name simplifies device installation for network integrators.
member-name	This can be a **datapoint** member or a block **property** member. To specify a property which applies to a datapoint member, use the datapoint member name and the property member name, separated by a period. Example: **nvoLoad.cpnLoadControl**
array-size	This attribute can only be applied to properties. The size supplied must be within the rules for array implementations detailed within the profile. If the size is not specified, the minimum array size specified by the profile is implemented.
initial-value	The initial value for the named member, expressed as a C language static initializer. You can specify any initial value, but the C compiler may report errors if the value provided does not meet the requirements of the type which is initialized. You can specify an **i-string** construct to simplify initialization of string-type members within a resource. For example, **i"room 101"** is equivalent to **{"room 101"}**.
event-handler	The name of your event handler function. The function must be a global function (not static) and match the IzoT Device Stack DX API requirements for the corresponding event. You do not need to specify the function prototype, but you must supply the implementation of this function.

Datapoint

The ***//@IzoT Datapoint*** directive creates one or more device datapoints. The Datapoint instruction syntax is as follows:

```
datapoint-declaration ::= datapoint-type variables ; //@IzoT Datapoint[(datapoint-modifiers)[datapoint-
attributes]
```

where

```
variables ::= variable | variable, variables
variable ::= name | name[number]
datapoint-modifiers :: datapoint-modifier | datapoint-modifier, datapoint-modifiers
datapoint-modifier  ::= direction-identifier | locator
locator ::= location="Izot Python Resources package path"
direction-identifier ::= direction=(Input | Output)
datapoint-attributes ::= datapoint-attribute | datapoint-attribute, datapoint-attributes
datapoint-attribute  ::= authentication(enabled= True | False [, configurable=  True | False])
        | comment("comment")
        | external(external-name)
        | init(initial-value)
        | onComplete(event-handler)
        | onUpdate(event-handler)
        | persistent(enabled=  True | False  [, configurable=  True | False ])
        | priority(enabled=  True | False  [, configurable=  True | False ])
        | service(servicetype=  ACKD | UNACKD_RPT | UNACKD [, configurable=  True | False])
```

The **authentication** attribute defaults to **enabled=False**, **configurable=True**.

The **persistent** attribute defaults to **enabled=False**, **configurable=True**.

The **priority** attribute defaults to **enabled=False**, **configurable=True**.

The **service** attribute defaults to **servicetype=ACKD**, **configurable=True**.

Datapoint Examples

```
UNVT_test a, b, c; //@IzoT Datapoint
SNVT_switch nviEnable; //@IzoT Datapoint(direction=Output)
```

Datapoint Syntax Elements

Syntax Element	Purpose
datapoint-type	The name of the datapoint type, e.g., **SNVT_switch**,**UNVT_lottery_numbers**.
variable	Name of the global C variable that implements this item.
initial-value	The initial value for the named member, expressed as a C language static initializer. You can specify any initial value, but the C compiler may report errors if the value provided does not meet the requirements of the type which is initialized. You can specify an i-string construct to simplify initialization of string-type members within a resource. For example, **i"room 101"** is equivalent to **{"room 101"}**.
event-handler	The name of your event handler function. The function must be a global function (not static) and match the IzoT Device Stack API requirements for the corresponding event. You do not have to specify the function prototype, but you must supply the implementation of this function.

Device Events

The **//@IzoT Event** directive registers one or more handlers for device-wide events. The Event instruction syntax is as follows:

```
event-declaration ::= //@IzoT Event[event-specs]
```

where

```
event-specs ::= event-spec | event-spec, event-specs
event-spec  ::= onOffline(event-handler)
        | onOnline(event-handler)
        | onReset(event-handler)
        | onWink(event-handler)
        | onService(event-handler)
```

Device Event Syntax Elements

Syntax Element	Purpose
Event-handler	The name of your event handler function. The function must be a global function (not static) and match the IzoT Device Stack API requirements for the corresponding event. You do not have to specify the function prototype, but you must supply the implementation of this function.

Property

The ***//*@IzoT Property** directive creates one or more device properties. The Property instruction syntax is as follows:

```
property-declaration ::= property-type variables; //@IzoTProperty[(locator)] [property-attributes]
```

where

```
variables ::= variable | variable, variables
variable  ::= name | name[number]
locator ::= location="Izot Python Resources package path"
property-attributes ::= property-attribute | property-attribute, property-attributes
property-attribute  ::= comment("comment")
        | flags(flags-value)
        | init(initial-value)
flags-value ::= flag-value | flag-value + flags-value flag-value ::= Const
        | DeviceSpecific
        | Disable
        | Manufacture
        | Offline
        | Reset
```

The flags attribute defaults to zero (no flag).

Property Examples

```
SCPTlocation cpLocation; //@IzoT Property init("Room  101")
SCPTnwrkCnfg configurationSource; //IzoT Property init(CFG_EXTERNAL), flags(Reset)
```

Tag

The ***//*@IzoT Tag** directive creates one or more messages. The Tag instruction syntax is as follows:

```
tag-declaration ::= IzotTag variables ; //@IzoTTag[tag-attributes]
```

where

```
variables ::= variable | variable , variables
variable ::= name | name[number]
tag-attributes ::= tag-attribute | tag-attribute , tag-attributes
tag-attribute  ::= bindable(True|False)
```

The bindable attribute defaults to True.

Tag Examples

```
SCPTlocation cpLocation; //@IzoT Property init("Room  101")
SCPTnwrkCnfg configurationSource; //IzoT Property init(CFG_EXTERNAL), flags(Reset)
```

Tag

The **//@IzoT Tag** directive creates one or more messages. The tag instruction syntax is as follows:

```
tag-declaration ::= IzotTag variables ; //@IzoTTag[tag-attributes]
```

where

```
variables ::= variable | variable , variables
variable ::= name | name[number]
tag-attributes ::= tag-attribute | tag-attribute , tag-attributes
tag-attribute  ::= bindable(True|False)
```

The bindable attribute defaults to True. Usage examples are as follows:

Tag Examples

```
IzotTag a, b;  //@IzoT Tag
IzotTag c;     //@IzoT Tag bindable(False)
```

Option

The **//@IzoT Option** directive selects one or more options. The Option instruction syntax is as follows:

```
option-declaration ::= Izot Option[option-attributes]
```

where

```
option-attribues ::= option-attribute | option-attribute , option-attributes
option-attribute ::= addresses (0..N)
        | aliases(0..N)
        | api(VALUE)
        | authentication(True | False)
        | buffersize(20...4096)
        | dynamic(blocks=0...N, datapoints=0..M)
        | explicit_addressing(True | False)
        | isi(True | False)
        | namespace(IDENTIFIER)
        | namestyle("v1" | "v2")
        | neutral(True | False )
        | output("output-filename")
        | programId("program-id")
        | property_policy("datapoint" | "file")
        | server(PATH)
        | servicebutton_held(0..31)
        | strict(True | False)
        | target("cpm-4000" | "cpm-4200" | "shortstack-classic" | "xif")
        | variable(name, value)
        | verbose(True | False)
```

Option Details

Option Attribute	Purpose
addresses	The number of address table entries to be allocated for the device. If not specified, the IzoT Interface Interpreter computes a number based on inspection of your application's interface. Target **xif** supports up to 4095 addresses, **cpm-4200** and **cpm-4000** targets support up to 254. This option is not supported with ShortStack, where the number of addresses is an intrinsic property of the chosen Micro Server that cannot be changed with IML.
aliases	The number of alias table entries to be allocated for the device. If not specified, the IzoT Interface Interpreter computes a number based on inspection of your application's interface. Target **xif** supports up to 8191 aliases, **cpm-4200** and **cpm-4000** targets support up to 127 aliases. This option is not supported with ShortStack, where the number of aliases is an intrinsic property of the chosen Micro Server that cannot be changed with IML.
api	This option selects API features as a combination of the following: **0** (the default) selects the basic API **1** adds local query and update functions **2** adds local utility functions. This option is only available with ShortStack.
authentication	Enables authentication for the device. The default is **False**. This option is only available with **cpm-4000** and **cpm-4200** targets.
buffersize	The requested minimum buffer size for all input and output buffers. The IzoT Device Stack allocates a buffer size equal to or larger than the requested number if possible, or fails to initialize if it cannot allocate the requested buffer size. If not specified, the IzoT Interface Interpreter allocates buffers according to the size of the largest datapoint (plus required protocol overhead). If a **Tag** object is declared, the IzoT Interface Interpreter ensures that the minimum buffer size is at least 255 bytes. Your explicit minimum buffer size request triggers warning 32 if it fails to meet the size required by the largest datapoint (and an allowance for protocol overhead). This warning becomes an error with the strict option.
dynamic	This option is supported with the **xif** target. The **dynamic** option accepts the number of dynamic blocks and datapoints to declare with the generated interface file. Both default to **0** (zero). The sum of dynamic and static datapoints cannot exceed 4096, and the number of dynamic blocks cannot exceed the number of dynamic datapoints. Example: `//@IzoT Option dynamic(blocks=100, datapoints=1000)`

explicit_addressing	This option accepts a Boolean True or False to enable or disable explicit addressing support. The default is **False.** This enables source addressing information for incoming messages and explicit addressing for outgoing application messages. Enabling this option enlarges the required data frame. This option is only available with ShortStack targets.
isi	This option accepts a Boolean True or False to enable or disable support for interoperable self-installation (ISI). The default is **False.** This option is only available with ShortStack targets.
namespace	This option specifies a namespace for use with runtime interface selection. It accepts an identifier that must match this regular expression: *[A-Za-z_][A-Za-z_0-9]** This option is only available with ShortStack targets.
namestyle	This option can be used to affect the generation of external names for datapoints, as reported in the interface file (XIF) and the device's self-identification data. Valid values are **"v1"** and **"v2"**. The default value is **"v2"**. • **namestyle "v1"** yields names that are compatible with earlier releases of IzoT Interface Interpreter • **namestyle "v2"** yields names that meet IzoT CT expectations, which results in a cleaner user interface when using automatically-generated datapoint names in a drawing Example: `//@IzoT Option namestyle("v1")`
neutral	This option accepts a Boolean True or False to enable or disable neutral output. The default is **False.** In neutral mode, the tool produces neutral time, data and version information in the generated output. This is useful when comparing output generated from different versions, as is the case during automated regression testing. This option can also be enabled with the IzoT Interface Interpreter command line option *--neutral*
output	The base name of the output files generated by the IzoT Interface Interpreter. The generated file names are **basename.c** and **basename.h**. The default base name is IzoTDev so the default output file names are **IzoTDev.c** and **IzoTDev.h** (for cpm-4000 and cpm-4200), and **ShortStackDev.c**, **ShortStackDev.h** for ShortStack. The option has no effect with target **xif**.
programId	The program ID for the device specified as a colon-separated string of hex digit pairs, e.g., **9F:FF:FF:00:00:00:00:01**.
property_policy	Specifies the global property implementation policy as one of **datapoint** or **file**. The option is supported for ShortStack and XIF targets.
server	Selects a ShortStack Micro Server. Example: `//@IzoT option server("microserver/standard/SS430_FT6050_SYS20000kHz")` The option is supported for ShortStack and XIF targets.
servicebutton_held	Specifies the service pin held notification interval, 0 to 31 seconds. The default is **0**. The option is supported with ShortStack targets.
strict	Changes some of the generated warnings to errors.
target	The target platform for the application, one of: c**pm-4000**, **cpm-4200**, **shortstack-classic**, or **xif**. Example: `//@IzoT option target("shortstack-classic")` Most IML options require that the target is specified before the option can be selected. It is recommended to declare the target option at the start of the IML declarations. Some installations of IzoT Interface Interpreter set the default value of this option to the target that matches the installation, but it is generally best to always declare the target. This ensures portability of the IML instructions between different installations. This option can also be set with the IzoT Interface Interpreter command line option *--target*.
variable	Defines or modifies a variable in the IzoT Interface Interpreter's environment. These variables are defined as a name, value pair, and may be referenced in some Interface Interpreter output using UNIX-style **$name** or **${name}** references.

verbose	Enables verbose IzoT Interface Interpreter output. The default is **False**. This option can also be selected with the IzoT Interface Interpreter command line option *--verbose*.