APPLICATION NOTE

AT08550: ZigBee Attribute Reporting

Atmel MCU Wireless

Features

- Details on ZCL Attribute Reporting with Atmel® BitCloud®
- Configuration and application usage of reportable attributes
- Automatic and Manual reporting
- Code snippets for ZCL Reporting based commands

Introduction

The ZigBee® Cluster Library specification (ZCL) defines cluster functionality as a common platform for developers of ZigBee applications. This is needed to create ZigBee devices that are inter-operable at the application level.

In the specification, one of the defined mechanisms to transfer cluster data is Attribute Reporting.

This application note is a user guide on the configuration and usage of Attribute Reporting in BitCloud SDK with references to the Home Automation application.

Table of Contents

1 Overview

The ZigBee Specification defines Application Profiles based on application domains such as Home Automation (HA) and Light Link (LL). These application profiles comprise the device descriptions (e.g.: Color Scene Remote Controller, Occupancy Sensor) and the set of clusters supported by the devices for that application domain.

The ZigBee Cluster Library (ZCL) specification [1] defines mandatory and optional clusters in terms of functional domains (e.g.: Lighting, Measurement, and Sensing).

1.1 ZCL Basic Terms

1.1.1 Cluster

A cluster is a related collection of attributes and commands which together define a communications interface between two devices, in the server-client model.

Example: Occupancy Sensing Cluster.

1.1.2 Attribute

An attribute is a variable or a set of variables that represent the functional state of the cluster. Attributes are characterized by a name, data type, range, access type (read/write), and report ability. Attributes are defined to be mandatory/ optional per cluster. Every attribute has an attribute ID defined by the ZCL specification [1].

Example: `occupancy` (a mandatory reportable attribute from the occupancy sensing cluster) with attribute ID 0x0000.

1.1.3 Command

Commands represent actions performed to manipulate the attributes of the cluster. Every command has an attribute ID defined by the ZCL specification [1].

Example: `identify query` (a mandatory command from the identify cluster) with attribute ID 0x01.

1.1.4 Cluster Server

A ZigBee logical device that stores the attribute values of a cluster.

Example: Occupancy Sensor is the server for the occupancy sensing cluster (mandatory as per [1]).

1.1.5 Cluster Client

A ZigBee logical device that affects or manipulates (read/write) the attributes of a cluster.

Example: Combined Interface is the client for the occupancy sensing cluster (mandatory as per [1]).

Note: The server and client definitions relate to the Profile Specification. Example: As per [2].

The definitions presented above are intended for quick reference. A detailed description of ZCL concepts is available in [1] and [4].

This application note discusses the Attribute Reporting feature which involves sending reportable attribute data from server to client without the client needing to request the values every time.

2 Attribute Reporting

In applications which require automatic, periodic update of reportable attribute data from server to remote clients, attribute reporting is a useful mechanism.

Consider a thermostat that needs to regulate temperature thresholds based on readings from a temperature sensor. Instead of the thermostat keeping track of the temperature values by sending explicit requests to the temperature sensor, periodic attribute reporting from the temperature sensor allows the thermostat to get regular temperature value updates.

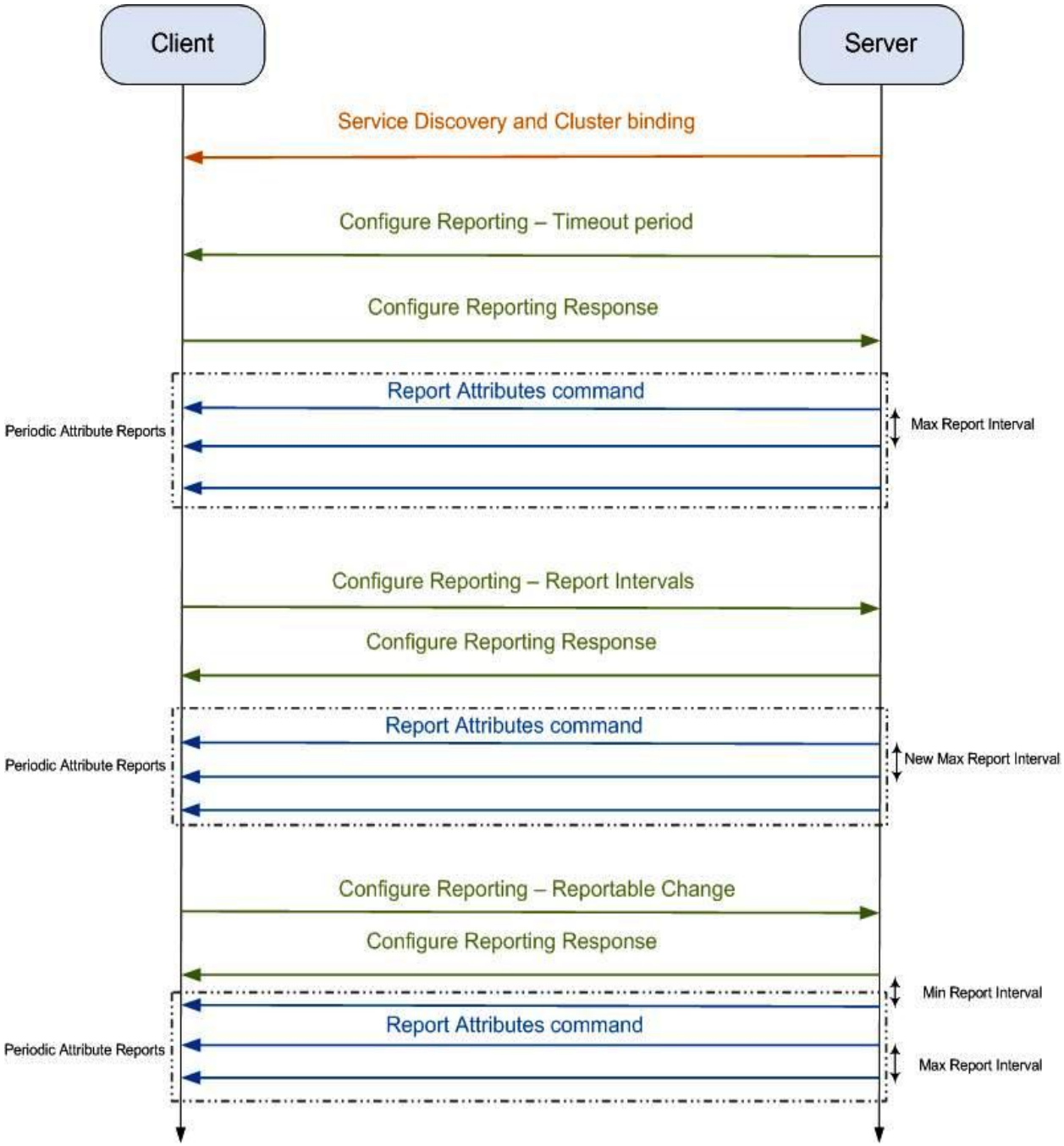
This is in contrast to the explicit request for attribute data from the cluster client (using the ZCL Read Attribute Request command). Attribute Reporting also allows a sleeping server device to update remote clients when it is awake.

Figure 2-1 shows the typical packet exchange for attribute report configuration and the actual periodic reports.

Attribute Reporting starts after a device has successfully joined a ZigBee network and completed service discovery. Service Discovery is the process by which a device in a ZigBee network identifies other devices that support similar services (clusters). After service discovery, the device knows the endpoints and addresses of devices supporting the same clusters as it does. The device, as a server cluster, can then send periodic reports to clients supporting the same cluster.

Service discovery and Binding concepts are described in detail in [4].

Figure 2-1. Attribute Reporting – Typical Packet Exchange Sequence



Note: Not all attributes in a cluster are reportable. The ZCL specification mentions whether reporting is mandatory for an attribute or optional or if reporting is not supported for a specific cluster and device.

2.1 Periodic Reporting - Types

BitCloud provides the application the flexibility to allow stack to automatically report attributes at user-configured intervals as well as to report attributes manually from the application.

2.1.1 Automatic Reporting

The user configures the reporting intervals of the cluster from the application at compile-time or at run-time. The BitCloud stack takes care of sending out the periodic reports once every **maximum reporting interval period**.

The stack uses the ZCL Report Attributes command to automatically send out periodic reports.

The various parameters to be set for automatic reporting are described in the sections below.

2.1.2 Reporting on Attribute Value Change

An attribute can be configured to be reported by the stack when the difference between its current value and previously reported one exceeds a certain threshold – so called ***reportable change*** parameter.

Reporting on change can be used when destination wants to be notified about specific changes in the attribute value that might occur fast and wouldn't be captured by periodic reports described above.

For example a thermostat might request a remote temperature sensor to send attribute reports when room temperature changes (in any direction) more than by 0.5 degree. Or a gateway device may need on-change reports of the On/Off attribute from a light to track when the light is turned off/on using another device (e.g. wall switch).

In some scenarios it is useful to limit amount of on-change reports to certain minimum time interval. This can be done using ***minimum report interval*** parameter. In such case if the difference in attribute value exceeds the reportable change, stack will hold on the transmission of the attribute report to ensure that the report is sent after the minimum report interval.

This is called on-change reporting and is described in Section 3.3.1.2.

Table 1-1. Reporting Behavior for Various Reporting Intervals and On-change Configurations

Min. report interval	Max. report interval	Reportable change	Behavior	Example
0	0	No	No reports	NA
0	0	Yes	Report only on change. No periodic reports.	Send report immediately when (current attribute value – last reported value) ≥ reportable change
Less than or equal to max. report interval. Can be zero.	Non-zero and greater than or equal to min. report interval	No	Periodic reports at maximum report interval.	Minimum Report Interval = 0 Maximum Report Interval = 12 A report is sent out every 12 seconds.
Less than or equal to max. report interval. Can be zero.	Non-zero and greater than or equal to min. report interval	Yes	Periodic reports at maximum report interval. Also: *Analog Data Type*: report on change after minimum report interval when (current attribute value – last reported value) >= reportable change value. *Discrete Data Type*: report on any change after minimum report interval. Reportable change parameter does not apply for discrete data kind.	Minimum Report Interval = 0 Maximum Report Interval = 12 Reportable Change = 5 Attribute value (current) = 13 Attribute value (previous) = 15 Attribute value (last reported) = 8 A report is sent out 12 seconds in once. (Analog/Discrete Data Type) Send report on any value change after minimum report interval.

Min. report interval	Max. report interval	Reportable change	Behavior	Example
Greater than max. report interval	Non-zero and less than min. report interval	No, Yes	Invalid configuration.	Minimum Report Interval = 14 Maximum Report Interval = 5 Continue reporting as per previous configuration of intervals.
Non-zero	0	No	No further periodic reports	Minimum Report Interval = 20 Maximum Report Interval = 0 Periodic reporting is stopped.
Non-zero	0	Yes	(Analog Data Type) Report on change after minimum report interval when (current attribute value – last reported value) ≥ reportable change. (Discrete Data Type) Send report on any value change after minimum report interval. No further periodic reports.	Minimum Report Interval = 20 Maximum Report Interval = 0 Periodic reporting is stopped. Reportable Change = 5 (Analog Data Type) Send report after 20 seconds, when the difference between current and last reported attribute value becomes ≥ reportable change (Discrete Data Type) Send report on any value change after minimum report interval.
0	0xFFFF	No, Yes	No further reports	No reports are sent even on change of attribute value.

Reportable Change, when set to zero, means that the minimum change to the attribute value is zero. This indirectly means that there is no change-based reporting.

2.1.3 Manual Reporting

In addition to automatic reporting at periodic intervals, the application can send out a report at any time using the `ZCL_AttributeReq()` function specifying the command ID to be `REPORT_ATTRIBUTES_COMMAND_ID`.

Section 3.3.2 provides a simple code snippet to send out a manual report from the application.

The application can choose to send the report periodically using an application timer or one-shot on a certain event occurrence. Details on application timer configuration are available in [4].

3 Implementing Attribute Reporting on a Cluster Server

BitCloud Developers Guide [4] describes the procedure of adding a new reportable attribute to a cluster. In this section, we cover the steps involved to setup periodic reporting of a reportable attribute from a cluster server to a client. The application taken as reference is the `HADevice` application (Home Automation) from BitCloud SDK [6] with multi-sensor device implementing server side of the occupancy sensing cluster.

Multi-sensor is a logical device implementation in BitCloud which can represent one sensor (such as occupancy sensor) or multiple sensors (among occupancy, humidity, illuminance, and temperature sensors). The application uses one endpoint for the multi-sensor and another separate endpoint for each supported sensor.

- This example focuses on a multi-sensor device with an occupancy sensor only

The Multi-sensor device is the server for the occupancy sensing cluster and the CI is the client.

3.1 Defining a Reportable Attribute

3.1.1 Cluster Header File

The attribute shall be defined as a reportable attribute using macro `DEFINE_REPORTABLE_ATTRIBUTE()`.

First, in the cluster header file in a macro that defines an instance of all cluster attributes, target attribute shall be defined using the macro:

```
DEFINE_REPORTABLE_ATTRIBUTE (name, props, aid, atype, min, max)
```

The attribute name (`name` argument) refers to the name of the attribute structure type defined lower in the file, attribute access (`props`), ID (`aid`) and data type (`atype`) are defined by ZCL specification, while `min` and `max` arguments are set by the application and correspond to reporting intervals, used as described in Section 3.2.1.

Example: In `BitCloud\Components\ZCL\include\zclOccupancySensingcluster.h`, occupancy attribute is defined as a reportable attribute and `occupansySensorType` is defined as non-reportable.

```
#define ZCL_DEFINE_OCCUPANCY_SENSING_CLUSTER_SERVER_ATTRIBUTES(min, max) \
  DEFINE_REPORTABLE_ATTRIBUTE(occupancy, ZCL_READONLY_ATTRIBUTE,
  ZCL_OCCUPANCY_SENSING_CLUSTER_OCCUPANCY_SERVER_ATTRIBUTE_ID,
  ZCL_8BIT_BITMAP_DATA_TYPE_ID, min, max), \
  DEFINE_ATTRIBUTE(occupancySensorType, ZCL_READONLY_ATTRIBUTE,
  ZCL_OCCUPANCY_SENSING_CLUSTER_OCCUPANCY_SENSOR_TYPE_SERVER_ATTRIBUTE_ID,
  ZCL_8BIT_ENUM_DATA_TYPE_ID)
```

- Also the attribute structure defined in the cluster header file must include additional reporting-related fields

Example: For the occupancy sensing attribute the `occupancy` structure is defined in `zclOccupancySensingcluster.h` file as follows:

A typical report attribute structure looks like:

```
struct PACK
{
    ZCL_AttributeId_t id;
    uint8_t type;
    uint8_t properties;
    uint8_t value;

    /* For internal use. keeps track of time to send next report */
    ZCL_ReportTime_t  reportCounter;

    /* application configured value */
```

```
    ZCL_ReportTime_t  minReportInterval;
    /* application configured value */
    ZCL_ReportTime_t  maxReportInterval;
    /* used for on-change reporting. Section 3.3.1.2 */
    uint8_t          reportableChange;
   /* max wait time between reports on the client. Section Error! Reference source not
found.*/
    ZCL_ReportTime_t  timeoutPeriod;
   /* For internal use in on-change reporting. Section 3.3.1.2 */
    uint8_t          lastReportedValue;
  } occupancy;
```

3.1.2 Application Code

If the cluster is specified as reportable in the cluster header file as described in Section 3.1.1, then the application can use corresponding macros to define and use such attribute. This procedure is explained in details in [4] and consists of following steps.

- Application shall define an instance of server attributes in the cluster source file. It also sets values for minimum and maximum reporting intervals.

Example: In `Applications\HADevice\multiSensor\src\msOccupancySensingCluster.c` file:

```
/* Global variable defined in application for supported occupancy sensing attributes */
ZCL_OccupancySensingClusterServerAttributes_t
osOccupancySensingClusterServerAttributes =
{ZCL_DEFINE_OCCUPANCY_SENSING_CLUSTER_SERVER_ATTRIBUTES(OCCUPANCY_SENSING_VAL_MIN_REP
ORT_PERIOD, OCCUPANCY_SENSING_VAL_MAX_REPORT_PERIOD)
};
```

- The cluster with the reportable attribute shall be part of the server cluster ID list. This list is used in the registering the device endpoint and for binding.

Example: In `Applications\HADevice\multiSensor\src\msClusters.c`:

```
/* list of supported server clusters */
ClusterId_t osServerClusterIds[OS_SERVER_CLUSTERS_COUNT] =
{
  OCCUPANCY_SENSING_CLUSTER_ID,
};

/* definition of supported server clusters */
ZCL_Cluster_t osServerClusters[OS_SERVER_CLUSTERS_COUNT] =
{
 DEFINE_OCCUPANCY_SENSING_CLUSTER(ZCL_SERVER_CLUSTER_TYPE,
   &osOccupancySensingClusterServerAttributes, NULL),
}
```

- Finally, application shall define a simple descriptor for the endpoint with server clusters

Example: In `Applications\HADevice\multiSensor\src\multiSensor.c`:

```
/* Occupancy Sensor Endpoint definition */
static ZCL_DeviceEndpoint_t osEndpoint =
{
  .simpleDescriptor =
```

```
  {
    .endpoint           = APP_SRC_ENDPOINT1_ID,
    .AppProfileId       = PROFILE_ID_HOME_AUTOMATION,
    .AppDeviceId        = HA_OCCUPANCY_SENSOR_DEVICE_ID,
    .AppInClustersCount  = ARRAY_SIZE(osServerClusterIds),
    .AppInClustersList   = osServerClusterIds,
    .AppOutClustersCount = 0,
    .AppOutClustersList  = NULL,
  },
  .serverCluster = osServerClusters,
  .clientCluster = NULL,
};
…
/*registering Occupancy Sensor Endpoint */
ZCL_RegisterEndpoint(&osEndpoint);
…
```

- In the cluster source file, define actions to be performed when stack notifies application that the reportable attribute parameters have been updated by the client, in an indication function

  In `Applications\HADevice\combinedInterface\src\ciOccupancySensingCluster.c`:

```
/* Pointer to cluster definition */
ZCL_Cluster_t *cluster = ZCL_GetCluster(APP_SRC_ENDPOINT_ID, OCCUPAN-
CY_SENSING_CLUSTER_ID, ZCL_CLUSTER_SIDE_SERVER);

/* If cluster is part of the supported server cluster list on the device endpoint, add
a function pointer to receive indications from stack when the attribute parameters are
updated. */
if (cluster)
   cluster->ZCL_AttributeEventInd = occupancySensorReportAttrEventInd;

/* indication function when reportable attribute parameters have been updated*/
static void occupancySensorReportAttrEventInd (ZCL_Addressing_t *addressing,
ZCL_AttributeId_t attributeId, ZCL_AttributeEvent_t event);
```

The application can use the event indication function to understand what action has been taken on the attribute parameters (read/write/configure attribute event/configure notification event), attribute ID and addressing. An example is presented in Section 4.3.

3.2 Configuring Reporting Parameters

This section describes how to configure the reporting parameters locally on the server device.

For remote configuration of reporting parameters from the client, please refer Section 4.5.

3.2.1 Application Definitions

- **Minimum and Maximum reporting intervals**

  The report intervals are in represented in terms of seconds.

  Compile-time settings of the reporting intervals shall be done in the application cluster header file.

  For the occupancy sensor, these are present in `msOccupancySensingCluster.h`

  ```
  #define OCCUPANCY_SENSING_VAL_MIN_REPORT_PERIOD 10
  #define OCCUPANCY_SENSING_VAL_MAX_REPORT_PERIOD 20
  ```

- **Reportable change**

  The reportable change parameter has the same data type as the attribute value data type. It is applicable only to analog data kind attributes.

  `LocalTemperature` attribute from the Thermostat cluster. Its range is from -237.15ºC to 327.67ºC with resolution 0.01ºC. If reportableChange is set to 4.5ºC, a report is sent after minimum report interval when: (current `LocalTemperature` – last reported value of `LocalTemperature`) ≥ ±4.5ºC.

Note: When changing reporting parameters such as report intervals and reportable change at run-time, update the report parameter values directly in the global variable and call `ZCL_StartReporting()` (Even if this function has been called earlier to start reporting, a local change in report parameter needs another call to this function).

Example: osOccupancySensingClusterServerAttributes.occupancy. minReportInterval = 5;

osOccupancySensingClusterServerAttributes.occupancy. maxReportInterval = 14;

(values in seconds)

- **Application level ZCL Command Buffers**

  These buffers are used by the command manager implementation that takes care of forming the `ZCL_AttributeReq()` and `ZCL_CommandReq()` using the available buffers.

  ```
  #define COMMAND_BUFFERS_AMOUNT 4
  ```

3.2.2 Configuration Server Parameters

Default values of CS parameters are available in `\ConfigServer\Include\csDefaults.h`.

If the developer wishes to change the default values, the parameter can be defined with the required value in application `configuration.h`.

- **ZCL Buffers**

  Buffers on the ZCL layer used for transmitting and receiving ZCL command frames. ZCL uses a quota system to divide the available number of buffers for outgoing command frames, incoming command frames, response frames and report frames. See `\ZCL\src\zclMemoryManager.c` for details.

  ```
  #define CS_ZCL_MEMORY_BUFFERS_AMOUNT 5
  ```

- **Binding table size**

  When the reporting happens between server and clients that are bound with each other, the binding table size limits the number of devices a report can be sent to.

  ```
  #define CS_APS_BINDING_TABLE_SIZE 10
  ```

3.3 Sending Attribute Reports

3.3.1 Automatic Reports by the Stack

3.3.1.1 Periodic Reports

As described in Section 2.1, an attribute can be configured for automatic reports by the stack. In such case the stack will send the attribute reports based on configured parameters to the nodes the cluster is bound to in the APS binding table.

From application, call API `ZCL_StartReporting()` once to start automatic reporting. It is recommended to call this API after service discovery and binding between server and client is complete.

Example: In `Applications\HADevice\multiSensor\src\multiSensor.c`, this is called in function `appEzModeDone()`.This is the callback indicating that the server-client binding has been completed.

3.3.1.2 On-change Reports

Apart from periodic reporting, it is possible to configure stack to transmit additional reports based on change of an attribute value. The client sends the `reportable change` parameter in the Configure Reporting command. The client sends this parameter representing the delta value between the current value and the last reported value of the attribute. The sign of the reportable change parameter is ignored, as per ZCL specification.

Note: The difference is calculated between the current attribute value and the last reported value (and not the immediate previous value of the attribute).

On the server, if the difference in attribute value exceeds the reportable change parameter received, an additional report is sent out after the minimum report interval period.

The `reportable change` parameter has the same data type as the attribute value data type. It is applicable only to analog data kind attributes.

Example: `LocalTemperature` attribute from the Thermostat cluster. Its range is from -237.15ºC to 327.67ºC with resolution 0.01ºC. If `reportableChange` is set to 4.5ºC, a report is sent after minimum report interval when:

(current `LocalTemperature` – last reported value of `LocalTemperature`) ≥ ±4.5ºC.

Every reportable attribute has a defined global variable in application. See Section 3.1.2 for an example.

To change the value of an attribute, application can:

- use `ZCL_WriteAttributeValue()` (or)
- change global variable struct object `.value` and call `ZCL_ReportOnChangeIfNeeded()`

Note: If the global variable is not changed, then reporting on change will not happen even if the above APIs are called.

`ZCL_ReportOnChangeIfNeeded()` shall be called whenever the application needs to send out an additional automatic report, apart from periodic reports.

Table 3-1 shows different scenarios where the application can use `ZCL_ReportOnChangeIfNeeded()`.

Table 3-1. Application Use Cases of `ZCL_ReportOnChangeIfNeeded()`

Scenario	Usage	Result
Periodic reporting is started after single call to `ZCL_StartReporting()`	(i) To send additional periodic reports, call `ZCL_ReportOnChangeIfNeeded()` at periodic intervals from application using an application timer	(i) Periodic reporting happens as per set maximum report interval. Additional reports are sent out every minimum report interval if application timer is less than minimum report interval or as per app timer duration if app timer duration > minimum report interval.
	(ii) To send additional reports based on external events, call `ZCL_ReportOnChangeIfNeeded()` from an external interrupt callback or any other event.	(ii) Periodic reporting happens as per set maximum report interval. When the external event occurs, an additional report is sent out after minimum report interval.

Note: `ZCL_ReportOnChangeIfNeeded()` can be used to send out an additional report only if either:

1. `ZCL_StartReporting()` has already been called from application.
2. Configure Reporting command packet with valid report intervals has been received from client.
3. The attribute is of discrete data kind as per ZCL specification.

Note: For automatic reports, there is no callback informing the application of the status of transmission on the server.

3.3.2 Manual Attribute Reports from Application

When the application wishes to send reports, the report packet can be formed in the application. Multiple attributes of the same cluster may be reported in the same report packet.

- Manual Reports of a reportable attribute can be sent in addition to periodic reports of the attribute by the stack
- Manual Reporting of non-reportable attributes (as per ZCL specification) is possible
- The application can decide the frequency of manual reports vis-à-vis the frequency of actual sensor measurements

Example: Sensor measurements obtained after ADC conversion are available via the configured ADC callback [4]. The actual sensor may be an analog sensor whose output is connected to one of the ADC input lines of the MCU.

The application can copy the ADC output data into the defined attribute buffer, form and send a manual report.

- The sensor data shall be copied into the value field of the attribute's global variable defined in the application (Section 3.1.2). An example is shown below.

```
/* global variable */
newSensorvalue = <adc_output_from_ADC_callback>;

/* To update the attribute value locally */
ZCL_WriteAttributeValue(APP_ENDPOINT_OCCUPANCY_SENSOR,
                                    OCCUPANCY_SENSING_CLUSTER_ID,
                                    ZCL_CLUSTER_SIDE_SERVER,
ZCL_OCCUPANCY_SENSING_CLUSTER_OCCUPANCY_SERVER_ATTRIBUTE_ID,
                                    ZCL_U8BIT_DATA_TYPE_ID,
                                    (uint8_t*)(& newSensorvalue));
```

Note: The attribute value can be of different data types as per ZCL specification and the data needs to be filled in as per the requirement.

- A manual report packet is made up of the general report structure (`ZCL_Request_t`) and one or more reportable attributes' data for one cluster, each of type `ZCL_Report_t`
- Define an application level buffer to hold the reportable attribute

```
/* Application buffer size */
#define OCCUPANCY_SENSING_ATTRIBUTE_BUFFER_SIZE 10

/* Application buffer to hold the attribute data */
static uint8_t osAttrBuf[OCCUPANCY_SENSING_ATTRIBUTE_BUFFER_SIZE];
```

- Define an instance of `ZCL_Report_t` to point to the defined application data buffer. The occupancy sensing cluster has only one reportable attribute. Hence the application buffer shall contain only a single report.

```
/* The report element for a reportable attribute shall point to the allocated application
buffer for that attribute */
ZCL_Report_t *reportAttrElement = (ZCL_Report_t*) osAttrBuf;

reportAttrElement->id = ZCL_OCCUPANCY_SENSING_CLUSTER_OCCUPANCY_SERVER_ATTRIBUTE_ID;

reportAttrElement->type = ZCL_U8BIT_DATA_TYPE_ID;
```

- Fill the report element attribute value with the sensor readings from the actual sensor

```
/* The report element value shall be the global variable value of the cluster's server
attributes */
reportAttrElement->value[0] = osOccupancySensingClusterServerAttributes.occupancy.value;
```

The struct member `value` serves as a variable-length array as per the attribute data type.

- Define an instance of the general report structure as shown below:

```
static ZCL_Request_t osAttrReq = {
  /* value defined as per ZCL specification */
  .id                     = ZCL_REPORT_ATTRIBUTES_COMMAND_ID,
  /* callback defined by app to get notified on status of packet transmission */
  .ZCL_Notify              = osReportNotify,
  /* Use APS_NO_ADDRESS as client's address has been added to local binding table during
discovery */
  .dstAddressing.addrMode     = APS_NO_ADDRESS,
  /* server endpoint */
  .endpointId               = APP_ENDPOINT_OCCUPANCY_SENSOR,
  /* server cluster ID */
  .dstAddressing.clusterId    = OCCUPANCY_SENSING_CLUSTER_ID,
  /* Profile ID */
  .dstAddressing.profileId    = PROFILE_ID_HOME_AUTOMATION,
  /* destination cluster side */
  .dstAddressing.clusterSide   = ZCL_CLUSTER_SIDE_CLIENT,
  /* destination end-point */
  .dstAddressing.endpointId    = APP_ENDPOINT_COMBINED_INTERFACE,
  /* report size */
  .requestLength            = sizeof(ZCL_Report_t),
  /* report payload copied into the application data buffer */
  .requestPayload           = osAttrBuf,
  /* ZCL header sub-field that specifies if a default response packet needs to be sent from
client on receiving the report packet. */
  .defaultResponse          = ZCL_FRAME_CONTROL_DISABLE_DEFAULT_RESPONSE,
};
/* To report, call the API below after binding is complete */
ZCL_AttributeReq(&osAttrReq);
```

3.4 Configuring the Reporting Timeout on a Remote Client

The reporting timeout value set on the client is the maximum wait time between reports after which client decides that there is a problem with reporting.

The server sends the Configure Reporting command to change the reporting timeout on the client. An example usage of this command is available in the HADevice reference application in BitCloud SDK.

When the client receives the configure report command from the server with the time-out period, the stack automatically sends the Configure Reporting Response command packet.

An indication is provided to application with attribute ID and event `ZCL_CONFIGURED_ATTRIBUTE_REPORTING_NOTIFICATION_EVENT` if the application has defined the call-back function `ZCL_AttributeEventInd()`.

When the server receives the Configure Reporting Response command packet from the client, the application is notified via configured callback function `ZCL_Notify()`.

A generic example implementation is presented below:

```
static void ZCL_ConfigureReportingResp(ZCL_Notify_t *ntfy)
{
  ZCL_NextElement_t element;
  ZCL_ConfigureReportingResp_t *configReportingResp;
  if(ZCL_SUCCESS_STATUS == ntfy->status && ZCL_ZCL_RESPONSE_ID  == ntfy->id)
  {
    /* Get the response payload using ZCL_GetNextElement */
    element.id          = ZCL_CONFIGURE_REPORTING_RESPONSE_COMMAND_ID;
    element.payloadLength = ntfy->responseLength;
    element.payload      = ntfy->responsePayload;
    element.content      = NULL;
    ZCL_GetNextElement(&element);

    configReportingResp = (ZCL_ConfigureReportingResp_t *) element.content;

    /* Configure Reporting Response command received from client */
    LOG_STRING(ntfyStatus, " +The config report response command was received with status
= 0x%x\r\n");
    appSnprintf(ntfyStatus, (unsigned) configReportingResp->status);
  }
}
```

4 Cluster Client Changes for Configuring and Receiving Reports

4.1 Cluster Header File

The cluster header file for the cluster that has the reportable attribute shall be included in the client side too.

In the case of occupancy sensor, it is `zclOccupancySensingcluster.h` as mentioned in Section 3.1.1.

4.2 Application Code

The cluster with the reportable attribute shall be part of the client cluster ID list. This list is used in the registering the device endpoint and for binding.

In `Applications\HADevice\combinedInterface\src\ciClusters.c`:

```
/* list of supported client clusters */
ClusterId_t  ciClientClusterIds[CI_CLIENT_CLUSTERS_COUNT] =
{
  BASIC_CLUSTER_ID,
  IDENTIFY_CLUSTER_ID,
  ONOFF_CLUSTER_ID,
  LEVEL_CONTROL_CLUSTER_ID,
  GROUPS_CLUSTER_ID,
  SCENES_CLUSTER_ID,
  OCCUPANCY_SENSING_CLUSTER_ID,
  TEMPERATURE_MEASUREMENT_CLUSTER_ID,
  HUMIDITY_MEASUREMENT_CLUSTER_ID,
  ILLUMINANCE_MEASUREMENT_CLUSTER_ID
};
/* definition of supported client clusters */
ZCL_Cluster_t ciClientClusters[CI_CLIENT_CLUSTERS_COUNT] =
{
  ZCL_DEFINE_BASIC_CLUSTER_CLIENT(),
  DEFINE_IDENTIFY_CLUSTER(ZCL_CLIENT_CLUSTER_TYPE, NULL, &ciIdentifyCommands),
  DEFINE_ONOFF_CLUSTER(ZCL_CLIENT_CLUSTER_TYPE, NULL, &ciOnOffCommands),
  DEFINE_LEVEL_CONTROL_CLUSTER(ZCL_CLIENT_CLUSTER_TYPE, NULL, &ciLevelControlCommands),
  DEFINE_GROUPS_CLUSTER(ZCL_CLIENT_CLUSTER_TYPE, NULL, &ciGroupsCommands),
  DEFINE_SCENES_CLUSTER(ZCL_CLIENT_CLUSTER_TYPE, NULL, &ciScenesCommands),
  DEFINE_OCCUPANCY_SENSING_CLUSTER(ZCL_CLIENT_CLUSTER_TYPE, NULL, NULL),
  DEFINE_TEMPERATURE_MEASUREMENT_CLUSTER(ZCL_CLIENT_CLUSTER_TYPE, NULL),
  DEFINE_HUMIDITY_MEASUREMENT_CLUSTER(ZCL_CLIENT_CLUSTER_TYPE, NULL),
  DEFINE_ILLUMINANCE_MEASUREMENT_CLUSTER(ZCL_CLIENT_CLUSTER_TYPE, NULL),
};
```

In `Applications\HADevice\combinedInterface\src\combinedInterface.c`:

```
/* Combined Interface Endpoint definition */
static ZCL_DeviceEndpoint_t ciEndpoint =
{
  .simpleDescriptor =
  {
    .endpoint          = APP_SRC_ENDPOINT_ID,
    .AppProfileId       = PROFILE_ID_HOME_AUTOMATION,
```

```
    .AppDeviceId        = HA_COMBINED_INTERFACE_ID,
    .AppInClustersCount  = ARRAY_SIZE(ciServerClusterIds),
    .AppInClustersList   = ciServerClusterIds,
    .AppOutClustersCount = ARRAY_SIZE(ciClientClusterIds),
    .AppOutClustersList  = ciClientClusterIds,
  },
  .serverCluster = ciServerClusters,
  .clientCluster = ciClientClusters,
};
```

4.3 Receiving Attribute Reports

In the cluster source file, define actions to be performed when a report is received from the server, in an indication function.

In `Applications\HADevice\combinedInterface\src\ciOccupancySensingCluster.c`:

```
/* Pointer to cluster definition */
ZCL_Cluster_t *cluster = ZCL_GetCluster(APP_SRC_ENDPOINT_ID, OCCUPAN-
CY_SENSING_CLUSTER_ID, ZCL_CLUSTER_SIDE_CLIENT);

/* If cluster is part of the supported client cluster list on the device endpoint, add a
function pointer to receive indications from stack when a periodic report from the server
is received. */
if (cluster)
 {
  cluster->ZCL_ReportInd = ciOccupancySensorReportInd;
  cluster ->ZCL_AttributeEventInd = ciOccupancySensorReportAttrEventInd;
 }

/* indication function when a report is received from server*/
static void ciOccupancySensorReportInd(ZCL_Addressing_t *addressing, uint8_t
reportLength, uint8_t *reportPayload);

/* indication function when report timeout has been updated by server*/
static void occupancySensorReportAttrEventInd (ZCL_Addressing_t *addressing,
ZCL_AttributeId_t attributeId, ZCL_AttributeEvent_t event);
```

The application can define the report indication callback function to take action when a periodic report is received from the server. An example implementation is shown below:

```
static void ciOccupancySensorReportInd(ZCL_Addressing_t *addressing, uint8_t reportLength,
uint8_t *reportPayload)
{
  ZCL_Report_t *rep = (ZCL_Report_t *)reportPayload;
  /* The attr value received in the periodic report from server is printed to serial console
*/
  LOG_STRING(reportAttrIndStr, "<-Occupancy Sensor Attr Report: t = %d\r\n");
  appSnprintf(reportAttrIndStr, (int)rep->value[0]);

  (void)addressing, (void)reportLength, (void)rep;
}
```

The client does not send a response packet for the Report Attributes command.

4.4 Reading the Reporting Configuration from a Remote Server

A client can read the current report configuration on a server such as the report intervals and reportable change value, using this ZCL command.

Example: A Home Automation Gateway device can read the reportable change value of the `CurrentHue` attribute of a color light to understand the delta change in Hue value that would cause an additional report to be sent from the color light.

When the Read Reporting Configuration command is received from the client, the stack on the server automatically prepares and sends the response.

On the client, to send the Read Reporting Configuration command, the code snippet below may be implemented in `ciOccupancySensingCluster.c` and a console command added for the combined interface.

```
void occupancySensingReadReportingConfig(APS_AddrMode_t mode, ShortAddr_t addr, Endpoint_t
ep,uint16_t attrID)
{
  ZCL_Request_t *req;
  ZCL_NextElement_t element;
  ZCL_ReadReportingConfigurationReq_t readReportingConfigReq;

  /* Try to get a free command buffer from the pool zclCommands[COMMAND_BUFFERS_AMOUNT] in
HADevice\common\src\commandManager.c */
  if (!(req = getFreeCommand()))
  return;

  /* Fill the read reporting configuration payload */
  readReportingConfigReq.direction  = ZCL_FRAME_CONTROL_DIRECTION_CLIENT_TO_SERVER;
  readReportingConfigReq.attributeId =
ZCL_OCCUPANCY_SENSING_CLUSTER_OCCUPANCY_SERVER_ATTRIBUTE_ID;

  /* Copy the payload into the ZCL request payload using  ZCL_PutNextElement() */
  element.payloadLength = 0;
  element.payload = req->requestPayload;
  element.id = ZCL_READ_REPORTING_CONFIGURATION_COMMAND_ID;
  element.content = &readReportingConfigReq;
  ZCL_PutNextElement(&element);

  /* Fill the ZCL request packet elements such as endpoint IDs and addressing info */
  fillCommandRequest(req, ZCL_READ_REPORTING_CONFIGURATION_COMMAND_ID, ele-
ment.payloadLength);
  fillDstAddressing(&req->dstAddressing, mode, addr, ep, OCCUPANCY_SENSING_CLUSTER_ID);

/* callback defined by app to get notified on receiving read reporting configuration response*/
  req->ZCL_Notify = ZCL_ReadReportingConfigResp;
  /* Use the command manager to send the Configure Reporting command to client */
  commandManagerSendAttribute(req);
}
```

When the read reporting configuration response is received from the server with the report parameters, the stack on the client processes the packet and the application is notified via configured callback function `ZCL_Notify()`. An example implementation is shown below.

```
static void ZCL_ReadReportingConfigResp(ZCL_Notify_t *ntfy)
{
  ZCL_NextElement_t element;
  ZCL_ReadReportingConfigurationResp_t *readReportConfigResp;
  ZCL_DataTypeDescriptor_t dataTypeDescriptor;

  if (ZCL_SUCCESS_STATUS == ntfy->status && ZCL_ZCL_RESPONSE_ID == ntfy-> id)
  {
    /* Get the response payload using ZCL_GetNextElement */
    element.id          = ZCL_READ_REPORTING_CONFIGURATION_RESPONSE_COMMAND_ID;
    element.payloadLength = ntfy->responseLength;
    element.payload      = ntfy->responsePayload;
    element.content      = NULL;
    ZCL_GetNextElement(&element);

    readReportConfigResp = (ZCL_ReadReportingConfigurationResp_t *) element.content;
  /* Configure Reporting Response command received from server */
   LOG_STRING(ntfyStatus, " +The current reporting configuration has been read with status
= 0x%x\r\n");
   appSnprintf(ntfyStatus, (unsigned)ntfy->status);

   /* Print the report intervals used on the server */
   LOG_STRING(RepParamStr, " +Max Report Interval = %d, +Min Report Interval = %d\r\n");
   appSnprintf(RepParamStr, (int)readReportConfigResp-> maxReportingInterval,
(int)readReportConfigResp-> minReportingInterval);

  /* Print the reportable change value, if applicable, used on the server */
  ZCL_GetDataTypeDescriptor(req->attributeType, req->reportableChange,
&dataTypeDescriptor);
  if (ZCL_DATA_TYPE_ANALOG_KIND == dataTypeDescriptor.kind)
  {
     uint8_t attrLength = ZCL_GetAttributeLength(readReportConfigResp -> attributeType,
(uint8_t *)& readReportConfigResp-> reportableChange);
   LOG_STRING(RepParamStr, "+Reportable change=\r\n");
    for(uint8_t index = 0; index <attrLength; index++)
    {
      LOG_STRING(valueStr, "%d\r\n");
      appSnprintf(valueStr, readReportConfigResp-> reportableChange[index]);
    }
   }
 }
}
```

Note: On the server, there is no notification to the application that the report parameters have been read by a client node.

4.5 Configuring the Reporting Parameters on a Remote Server

The client can configure the report parameters such as the report intervals and the reportable change on a server. The Configure Reporting command is used for this purpose and a single command packet may contain multiple records for different reportable attributes in a cluster.

5 Setting up HADevice Application for Attribute Reporting

Setting up report configuration and periodic reporting in the HADevice application in BitCloud SDK [6] is as per Figure 2-1.

- Compile and load the application images on the server (occupancy sensor) and the client (combined interface). Both devices have a serial console as user interface as mentioned in [5].
- Power up the Combined Interface and wait for 180 seconds (`EZ_MODE_INTERVAL`). This is needed because, by default, both the CI and the multi-sensor are Ez-Mode initiators. The Ez-mode Commissioning process requires an initiator and a target.
- After `EZ_MODE_INTERVAL` is complete, set CI as Ez-Mode target using console commands:

```
/* console command to set CI as Ez-Mode target */
setEzModeType 0

/* console command to start Ez-Mode on CI */
startEzMode
```

- Power up the multi-sensor, it joins the network, and sends an identify query request. The CI responds and the multi-sensor does service discovery and device discovery with the CI.
- The occupancy sensor sends a Configure Reporting command (Section 4.5) to the CI with timeout period set as twice the maximum report interval. The CI sends a Configure Reporting response command with status.
- The occupancy sensor, then, starts sending periodic reports to the CI

5.1 Storing Reportable Attributes in NVM

It is possible to store reportable attribute values in non-volatile memory using PDS APIs as described in [4].This is useful for retaining values over power failure or reset events.

For the occupancy sensing cluster attribute `occupancy`, the memory ID is defined in ms`Pdt.c` in `HADevice` application.

```
PDS_DECLARE_FILE(APP_MS_OCCUPANCY_MEM_ID,
sizeof(osOccupancySensingClusterServerAttributes.occupancy),
&osOccupancySensingClusterServerAttributes.occupancy,  NO_FILE_MARKS);
```

where,

`osOccupancySensingClusterServerAttributes` is the global variable defined in application for occupancy sensing attributes as per Section 3.1.2.

The memory ID is mapped to an application parameter memory ID set aside in PDS implementation for application storage. This mapping can be found in `appConsts.h`.

```
#define APP_MS_OCCUPANCY_MEM_ID      APP_PARAM1_MEM_ID
```

Storing can be done in application using the `PDS_Store()` API.

Example: `PDS_Store(APP_MS_OCCUPANCY_MEM_ID);`

In the `HADevice` application, during initialization after power failure, restoration from NVM to RAM is performed as shown below.

```
if (PDS_IsAbleToRestore(APP_MS_OCCUPANCY_MEM_ID))
  PDS_Restore(APP_MS_OCCUPANCY_MEM_ID);
```

6 References

[1] ZigBee Cluster Library Specification (075123r04ZB).

[2] ZigBee Home Automation Profile Specification.

[3] ZigBee Light Link Profile Specification.

[4] AVR2050: BitCloud Developers Guide.

[5] AVR2052: BitCloud Quick Start Guide.

[6] BitCloud SDK.

7 Revision History

Doc Rev.	Date	Comments
42334A	01/2015	Initial document release.

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**