

AUTOMATIONS DOCUMENTATION

REFERENCE MANUAL



Table of Contents

I: Automation template	1
1. Overview	2
2. Automations overview	3
3. Guidelines for automation template bundle	5
4. bundle-info.json	6
4.1. id	6
4.2. category	6
4.3. keywords	7
4.4. version	7
4.5. publisher	7
4.6. publisher-id	7
4.7. name	7
4.8. description	8
4.9. bundle-html	8
4.10. item-description-html	8
4.11. bundle-pdf	8
4.12. logic	8
4.13. configuration-schema	9
5. bundle-html	11
6. Automation JavaScript file	12
7. Configuration schema	14
7.1. Attributes	15
7.1.1. \$schema	15
7.1.2. \$id	15
7.1.3. ubisys::presentation::template::title	15
7.1.4. ubisys::presentation::template::icon-pdf	16
7.1.4.1. type	16
7.1.5. properties	16
7.1.6. Data type and supporting attributes	16
7.1.6.1. type	16
7.1.6.2. default	17
7.1.6.3. minimum	17
7.1.6.4. maximum	17
7.1.6.5. minItems	17
7.1.6.6. maxItems	17
7.1.6.7. ubisys::type	17
7.1.7. name	18
7.1.8. description	18
7.1.9. ubisys::short-description	18
7.1.10. ubisys::presentation::order	19

7.1.11. ubisys::presentation::placeholder	19
7.1.12. ubisys::presentation::key	20
7.1.13. ubisys::presentation::mandatory	20
7.1.14. ubisys::presentation::visible	20
7.1.15. ubisys::value::cross-reference	20
7.1.16. ubisys::application-filter	20
7.1.16.1. qualifying-device-types	21
7.1.16.2. disqualifying-device-types	21
7.1.16.3. qualifying-clusters	21
7.1.16.4. include-groups	21
7.1.17. uniqueltems	21
7.1.18. ubisys::value::full-scale	22
7.1.19. ubisys::value::stepping	22
7.1.20. items	22
7.1.21. ubisys::value::options	23
7.2. Examples	23
7.2.1. Properties examples using type	23
7.2.1.1. String attribute	23
7.2.1.2. Array attribute	24
7.2.1.3. Integer attribute	24
7.2.1.4. Number attribute	25
7.2.1.5. Boolean attribute	25
7.2.1.6. Object attribute	25
7.2.2. Properties examples using ubisys::type	26
7.2.2.1. zigbee-application-instance attribute	26
7.2.2.2. time-of-day attribute	26
7.2.2.3. color-temperature-k attribute	27
7.2.2.4. percentage attribute	27
7.2.2.5. cross-reference attribute	27
7.2.2.6. lightlevel:lux attribute	27
7.2.3. "Motion-based lighting control system" use case	29
8. Adding support for other languages	34
9. index.plist	36
10. Parsed JSON file	37
11. Troubleshooting_errors	38
12. Integrated icons	40
II: Javascript	78
13. Features	79
14. Fundamentals	80
14.1. Naming conventions	80
14.1.1. Class	80
14.1.2. Interface	80

14.1.3. Inheritance	81
14.2. Firmware version references	81
14.3. APIs/Interfaces	81
14.4. Zigbee and Zigbee Cluster Library (ZCL) Basics	81
15. Programming Interface	83
15.1. Globals and Builtins	83
15.2. Timer API	83
15.2.1. Exported functions	83
15.2.1.1. createTimer()	83
15.2.1.2. createPeriodicTimer()	84
15.2.1.3. createDayTimer()	84
15.2.2. Interfaces and Classes	85
15.2.2.1. Timer	85
15.3. Zigbee API	86
15.3.1. Exported functions	86
15.3.1.1. onReady()	86
15.3.1.2. onUpdate()	86
15.3.1.3. getDevice ()	87
15.3.1.4. getGroupByName()	87
15.3.1.5. getGroupById()	87
15.3.1.6. getGroupByAddress()	88
15.3.1.7. getScene()	88
15.3.1.8. getSceneById()	88
15.3.1.9. getScenes()	89
15.3.1.10. lookup()	89
15.3.2. Interfaces and classes	90
15.3.2.1. Device	90
15.3.2.2. Application	91
15.3.2.3. Group	93
15.3.2.4. Scene	94
15.3.2.5. Cluster	95
15.3.2.6. OnOffCluster	99
15.3.2.7. LevelControlCluster	100
15.3.2.8. WindowCoveringCluster	102
15.3.2.9. ClientCluster	105
15.3.2.10. Attribute	107
15.3.2.11. ZigbeeStatus	108
15.3.3. Cached Attributes	108
15.3.3.1. Power Configuration Cluster (0x0001)	108
15.3.3.2. On/Off Cluster (0x0006)	109
15.3.3.3. On/Off Switch Configuration (0x0007)	109
15.3.3.4. Level Control Cluster (0x0008)	109

15.3.3.5. OTA Upgrade (0x0019)	109
15.3.3.6. Poll Control (0x0020)	110
15.3.3.7. Door Lock (0x0101)	111
15.3.3.8. Window Covering Cluster (0x0102)	111
15.3.3.9. Thermostat (0x0201)	112
15.3.3.10. Fan Control (0x0202)	113
15.3.3.11. Color Control Cluster (0x0300)	113
15.3.3.12. Ballast Configuration (0x0301)	114
15.3.3.13. Illuminance Measurement (0x0400)	114
15.3.3.14. Illuminance Level Sensing (0x0401)	114
15.3.3.15. Temperature Measurement (0x402)	114
15.3.3.16. Relative Humidity Measurement (x0405)	114
15.3.3.17. Occupancy Sensing (0x0406)	114
15.3.3.18. Leaf Wetness (0x0407)	115
15.3.3.19. Soil Moisture (0x0408)	115
15.3.3.20. IAS Zone Cluster (0x0500)	115
15.3.3.21. IAS WD Cluster (0x0502)	115
15.3.3.22. Metering (0x0702)	115
15.3.3.23. Device Setup (0xfc00)	116
15.3.3.24. Dimmer Setup (0xfc01)	116
15.4. Buffer Support	116
15.5. Push Notification API	117
15.6. Global variables	117
15.7. HTTP Client API	119
15.7.1. Exported functions	119
15.7.2. Classes	119
15.7.2.1. HttpRequest	119
15.7.2.2. FormData	123
15.7.3. HttpResponse	123
15.8. HTTP Server API	123
15.8.1. Exported functions	123
15.8.2. Classes	124
15.8.2.1. HttpRequest	124
16. Examples	126
16.1. Toggle a light every 5s seconds	126
16.2. Control a light via a motion detector	126
16.3. Write the StartUpOnOff attribute in the On/Off cluster	127
16.4. Control a light via an HTTP handler	128
16.5. Send an HTTP request on button press	128
17. Revision History	130
18. Contact	131

I: Automation template

1. Overview

Reproduction and copies (also in extracts) only with the consent of ubisys technologies GmbH. This document may contain errors in content. However, it will be revised regularly and corrected accordingly in the next edition. We accept no liability for errors in content. Changes in the sense of technical progress can be made without prior notice.

Copyright© 2020-2021 ubisys technologies GmbH, Düsseldorf, Germany.
All rights reserved.

2. Automations overview

Automations is one of the key features in any smart IoT application. With ubisys products, one has a variety of ways to define the automations and use their IoT products flexibly. Utilizing user-defined scripts and templates, is one of the smart way to define such automations. A developer or an end user with a basic knowledge of JavaScript and JavaScript Object Notation (JSON) can create such automations and use them with the ubisys app. These automations are template based hence they can be easily configured, and yet they are enormously versatile and sophisticated. A typical example of such an automation is the regulation of luminaires based on motion sensors and naturally incident daylight. Developers can define filter properties to group devices that support motion sensing and another group for luminaires. Similarly, one can also define other properties like threshold values and can expose it to the user interface to make it configurable in runtime. Once the properties are defined, the developer can then create a JavaScript file to implement the actual logic of the automation script which completes this template.

Once the template is ready, it is parsed through the ubisys app and a JSON message is created, which is then directly forwarded to the ubisys gateway without any cloud round-trips. With the help of such templates, a user interface for configuring all automation parameters can be created for the end-user significantly helping to hide the complexity of a script language. Such templates are then made available in form of bundle file in the .jsb format, which stands for "JavaScript bundle". Apart from the ubisys smart home app (currently iOS, Android will follow), these automations can also be deployed via custom cloud integration (push automation template from a cloud service).

This document is a developer guide on how to create customizable automation templates. In the first chapter, the overall bundle structure and the supporting files are explained, followed by supported attributes required to configure the automation. Lastly, a use case of a motion based lighting control system is discussed. The reader of this document is expected to have a basic knowledge of JSON and JavaScript programming.

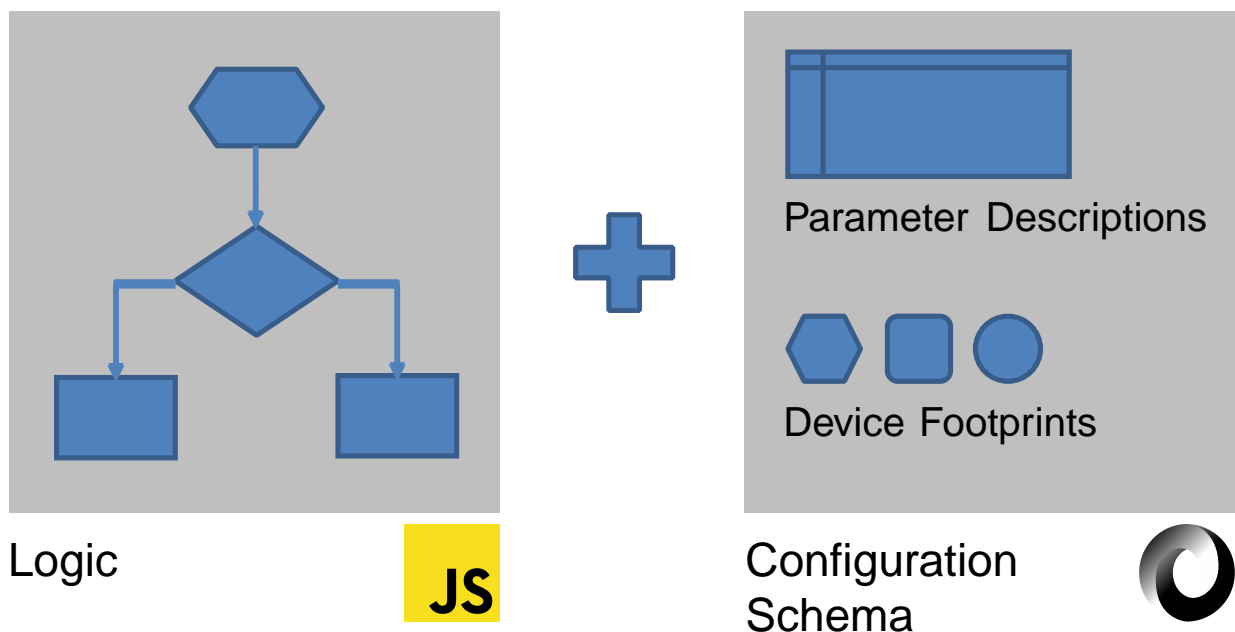


Figure 1. Automation template overview

The complete template can be divided into two sections. One that describes the logic which is the backbone of this template and the other one for configurations. To define the automation logic, JavaScript language is used. There must be at least one .js file in the automation template bundle

describing the expected logical actions. Different device footprints and parameters must be defined and configured by means of configuration schema files. For this purpose, the JSON format is used. Such a configuration.schema.json file includes system-wide parameters, automation properties that can also be made available to configure via the user interface. Thus, the logic along with the configuration schema contributes to the automation template. The details of these files are discussed in following sections.

3. Guidelines for automation template bundle

This chapter provides guidelines on creating a new automation template and running it in the ubisys iOS application. Developers are recommended to follow these steps and add further customization based on requirements.

1. Define the idea behind the automation and note down the trigger events and expected logical operations to execute the automation.
2. Create a JSON file named `bundle-info.json` and add basic details of the automation template. The name of the files defined in further steps should be exactly the same as described in this file. In depth details are explained in [Chapter 4](#).
3. Add an icon image for the automation in pdf format. In depth details are explained in [Section 4.11](#)
4. Create an HTML file to describe the logic behind the automation. Developers can also add images, charts to make it more descriptive. See [Chapter 5](#) for more details.
5. Create a JavaScript file to define the core logic of the automation. Define the functions to perform the logical operations based on the trigger events as defined in step one. Declare the properties which can be exposed to the user interface for further configuration. Developers can also add additional JavaScript files based on requirements. More details are available in [Chapter 6](#).
6. Create a configuration schema JSON file to define these properties and their respective parameters. Developers can use icons supported by the ubisys app or they can also add customized icons. More details are available in [Chapter 7](#).
7. To add support for additional languages (if required), add respective patch files as described in [Chapter 8](#).
8. Once all files are ready, zip the files and rename the zipped file same as "id" in `bundle-info.json` and extension as `.jsb`
9. Create a file named `index.plist` that includes basic details of the automation bundle file. More details are available in [Chapter 9](#)
10. Upload the bundled and plist file on the web-server and note down the link.
11. To test/run the automation, provide this link in the third-party repository section of the automations tab in the ubisys app and test the automation.

The minimal structure of the automation bundle should follow the structure as shown in the table below.

Table 1. Files in automation template bundle

File	Requirement
<code>bundle-info.json</code>	Mandatory
<code>bundle-pdf (bundle_image_icon.pdf)</code>	Mandatory
<code>bundle-html (bundle_information.html)</code>	Mandatory
<code>logic (automation_script.js)</code>	Mandatory
<code>configuration-schema (configuration.schema.json)</code>	Mandatory
... further automation files (scripts, icons, etc.)	Optional

An example explaining these steps is available at [Section 7.2.3](#).

4. bundle-info.json

The information of the complete automation bundle is summarized in the bundle-info.json file. Details such as name, version number, description etc. are stored here, as well as defining logic script files, description files, icons, etc. Each of these attributes have a unique significance and must be defined based on the requirements from the automation template. The overall structure must be in JSON format and the language used must be English. To add support for other languages, the developer needs to create a patch file for every supported language, which includes equivalent text. See [Chapter 8](#) for more details. The typical structure of the bundle-info file is as shown below.

```
1 {
2   "id": "lighting.example",
3   "category": "lighting",
4   "keywords": [ "illuminance", "sensors", "HCL"],
5   "version": "1.0.0",
6   "publisher": "Organisation name",
7   "publisher-id": "organisation.id",
8   "name": "Lighting Control Script",
9   "description": "Lighting control script for home automation",
10  "bundle-html": "bundle_information.html",
11  "item-description-html": "item_description.html",
12  "bundle-pdf": "bundle_image_icon.pdf",
13  "logic": "automation_script.js",
14  "configuration-schema": "configuration.schema.json"
15 }
```

The details of each of these attributes are discussed below.

4.1. id

Mandatory

Id is the unique identifier of the automation template with which the developer can distinguish different templates from another. The data type must be string and the contents cannot be empty.

```
1 "id": "lighting.example"
```

4.2. category

Mandatory

Category is the domain identifier of the given template for example, "lighting", "energy", "lifestyle" and must be of string data type. In the current ubisys application, there are no any restrictions on selecting the category, but the developer is recommended to use the best suited phrase for their automation template.



As of now, the ubisys application does not support filtering templates using category.

```
1 "category": "lighting"
```

4.3. keywords

Mandatory

A set of keywords which help users find a suitable template using a keyword search. It must be defined as an array of strings.



As of now, the ubisys application does not support keyword search functionality.

```
1 "keywords": [ "illuminance", "sensors", "HCL" ]
```

4.4. version

Mandatory

Version is used as an identifier that can be used to show the current version of your automation template, that is also used to display information about the template on the user interface, see [\[Automation_template_main_UI\]](#) for more details.

```
1 "version": "1.0.0"
```

4.5. publisher

Optional

Name of the owner of the automation template, for example name of the company, in string data type. This attribute is currently not used for any user interface, but the developer can provide this information for future reference.

```
1 "publisher": "Organization name"
```

4.6. publisher-id

Optional

Shows the publisher-id of the owner of the automation template, in string data type. The developer can specify an ID specific to his organisation. This attribute is currently not used for any user interface, but developer can provide this information for future reference.

```
1 "publisher-id": "organization.id"
```

4.7. name

Mandatory

The name of the automation template that describes the actual objective of this template. The name attribute appears on the user interface as seen in the [\[Automation_template_main_UI\]](#).

```
1 "name": "Lighting Control Script"
```

4.8. description

Mandatory

A short description to provide more details about the automation template must be provided with this attribute. See [\[Automation_template_main_UI\]](#) for more details.

```
1 "description": "Lighting control template for home automation"
```

4.9. bundle-html

Mandatory

Filename of an HTML file, with full information about features, logic, etc. This file may also include further references to CSS and JavaScript files in the bundle. With this, the developer can create a detailed explanation of the automation template in the form of an HTML document, which then will be parsed and displayed on the user interface. See [\[Automation_template_main_UI\]](#) for more details.

```
1 "bundle-html": "bundle_information.html"
```

4.10. item-description-html

Optional

Name of an HTML template file, with a placeholder for a plain text string that is attached to a parameter via the JSON schema.

```
1 "item-description-html": "item_description.html"
```

4.11. bundle-pdf

Mandatory

Filename of an icon in the PDF format which will be shown next to the name of the automation template and automation instances. See [\[Automation_template_main_UI\]](#) for more details.

```
1 "bundle-pdf": "bundle_image_icon.pdf"
```

4.12. logic

Mandatory

The most important field in this JSON is the logic that should point to the main automation JavaScript file to be executed on the ubisys Gateway by the Duktape JavaScript engine. The logic file utilizes the properties defined in the schema file and define the automation logic. See [Chapter 6](#) for more details.

```
1 "logic": "automation_script.js"
```

4.13. configuration-schema

Mandatory

The configuration-schema file holds all the possible configurable properties for the given automation template. The schema files defines the data type, boundary values, default values, icons etc. of all given properties as well as allowing the developer to define how it should be represented on the user interface. See [Chapter 7](#) for more details.

```
1 "configuration-schema": "configuration.schema.json"
```

Once the JSON and HTML files are parsed, the UI will be generated as shown.



Figure 2. Automation template main UI

5. bundle-html

The bundle-html is one of the most vital documents explaining the logic behind the automation template to the user. Once the template is selected from the user interface, an overview of the automation template along with the description will be loaded on to the user interface (See [\[Automation_template_main_UI\]](#) for more details). The developer can include a variety of images and texts in sections, subsections to make it intuitive so that end-user can understand the automation. The name must be the same as defined in the [Section 4.9](#) attribute from bundle-info. The structure of this file must be in standard HTML.

```
1 <html>
2   <head>
3     <!-- ... Include meta files if any -->
4     <meta http-equiv="Content-Type" content="text/html" />
5     <style>
6       /* ... Style formatting for header, div, body, etc */
7       h1 { color: black; font-size: 12px; font-weight: bold; }
8       p { color: gray; font-size: 10px; font-weight: normal; }
9     </style>
10  </head>
11  <body>
12    <div id="content">
13      <!-- ... Include images
14         ... Description
15         ... More description -->
16    <p></p>
17    <h1>Functions</h1>
18    <p>This automation template provides smart ways to control
19      your devices.</p>
20    </div>
21    <script>
22      /* ... Script if required */
23    </script>
24  </body>
25 </html>
```


6. Automation JavaScript file

The automation script contains the core logic part of the automation template. The name must be the same as defined in the "logic" attribute from bundle-info [Section 4.12](#). The JavaScript Runtime provides a way to extend your ubisys smart home system with custom logic through such user-defined scripts, written in the popular JavaScript language. It was chosen as the language for custom, user-defined automation in the ubisys smart home system, due to its flexibility, widespread use and knowledge. Add to this the fact that it is fairly uncomplicated to learn for beginners while also providing ample power for advanced users.

As this JavaScript file defines the logic, it does not define or interacts with the user interface. All the properties defined in the configuration-schema are parsed by the ubisys application and the resultant configuration JSON file is generated. See for [Chapter 10](#) more information. These properties are then available as global variables to the functions in this file. Please make sure to use the identical name for all the properties in the schema and JavaScript files. Following is a typical overview of the logic JavaScript file.

```
1 ///////////////////////////////////////////////////////////////////
2 // Global variables
3
4 // For Zigbee apis
5 var Zigbee = require('sys/zigbee');
6
7 ///////////////////////////////////////////////////////////////////
8 // The configuration is supplied to this script by the ubisys Zigbee
9 // JavaScript Engine. Its contents will be created by end users using the
10 // ubisys mobile application. These configurations were defined in the
11 // configuration.schema.json file. Lets say, there are 3 properties defined
12 // property_1, property_2, property_3
13
14 var config = instanceConfiguration();
15
16 ///////////////////////////////////////////////////////////////////
17 // Functions
18
19 // Executes the automation logic
20 function runAutomation()
21 {
22     if (config.property_1)
23     {
24         config.property_2.turn_On();
25     }
26     config.property_3.run();
27 }
28
29 // Main function
30 Zigbee.onReady(function()
31 {
32     runAutomation();
33 });
```

This example showcases the basic skeleton of the automation JavaScript file. The developer needs to use the properties defined in the configuration schema and define the functional logic in a similar way to gain the expected logical output. The current automation template supports following features:

- Timers
 - Single-shot and periodic timers
 - Day timers and solar timers (at or relative to sunrise/sunset)
- Zigbee Integration
 - Generic support for all clusters and cluster-directed commands
 - Specialized support for the On/Off, Level Control and Window Covering clusters
 - Support for a subset of attribute types

A simple use case of a motion based lighting control system is available in [\[motion_based_lightning_use_case_section\]](#). For more details regarding the use of these features and its configurations, refer to the smart home JavaScript Runtime Reference Manual.

7. Configuration schema

This file describes all configurable parameters of the automation template, them being type, title, icon, as well as vital properties. The name must be same as defined in the [Section 4.13](#) attribute from bundle-info. Please note that the properties mentioned in this schema file only allows the developer to expose these configurations to the user interface. The developer needs to make sure that these properties are being used in the automation logic file so as to perform any action/reaction based on the automation template requirements.

The language of the text used in configuration schema must be English. However, the developer can add support for other languages by providing additional patch.schema.json files which will be then used to update the localized JSON object of the configuration schema and hence the user interface as well. See [Chapter 8](#) for more details about localization. The typical structure of a schema document looks as follows

```
1 {
2   "$schema": "http://json-schema.org/schema#",
3   "$id": "http://your.organisation.com/configuration.json",
4   "ubisys::presentation::template::title": "More Settings...",
5   "ubisys::presentation::template::icon-pdf": "#automation.pdf",
6   "type": "object",
7   "properties":
8     {
9       "... Template properties"
10    }
11 }
```

The schema file starts with the schema tag defining the schema structure used in this file, followed by the id of the configuration schema. The title and icon files are used for user interface and must be configured based on your automation template requirements. In the properties, the developer must define as well as configure different device footprints, parameters and attributes which are required in the automation logic JavaScript file. These attributes can be initialized to a default value or can be made available on the user interface for further customization. The data type of each variable must be in accordance with the way they are used in the automation JavaScript file, but the overall structure of the file should be in JSON format.

Within the properties section all customizable parameters must be defined. Here in this section, an example of the system level property is discussed to provide a broader perspective of use of properties. The details of attributes used to create such properties are discussed in following subsections of this document.

```

1 "properties":
2 {
3   "primary_level_value" :
4   {
5     "type": "integer",
6     "name": "Primary Level Value",
7     "default": 178
8     "minimum": 1,
9     "maximum": 254,
10    "description": "<p>Initial value of brightness</p>",
11    "ubisys::short-description": "Initial value"
12  }
13 }

```

The given example shows the declaration of a property named "primary_level_value" of the integer data type with some default value. Again for the UI purpose, descriptions can be added along with minimum/ maximum limits. As such there can be one or more properties based on your automation template requirements.

7.1. Attributes

7.1.1. \$schema

Mandatory

This is the reference schema structure used to create the JSON file. It must have one of the standard url paths. For example

```

1 "$schema": "http://json-schema.org/schema#"

```

7.1.2. \$id

Mandatory

The unique ID of the configuration schema file is maintained in this attribute. It may have a link to the manufacturer specific file for a customised web URL for more reference. For example

```

1 "$id": "http://your.organisation.com/your-automation-configuration.json"

```

7.1.3. ubisys::presentation::template::title

Mandatory

This attribute must have the title of the template section. The value must be in the form of a string. See [Figure 3](#) for more information.

```

1 "ubisys::presentation::template::title": "More Settings ..."

```

7.1.4. ubisys::presentation::template::icon-pdf

Mandatory

The icon of the template is configured via this attribute. The developers can use application-provided icon files with "#" prefix, or provide their own icon file in the PDF format. This icon is then parsed and displayed on the user interface. See [Figure 3](#) for more information.

```
1 "ubisys::presentation::template::icon-pdf": "settings.pdf"
```

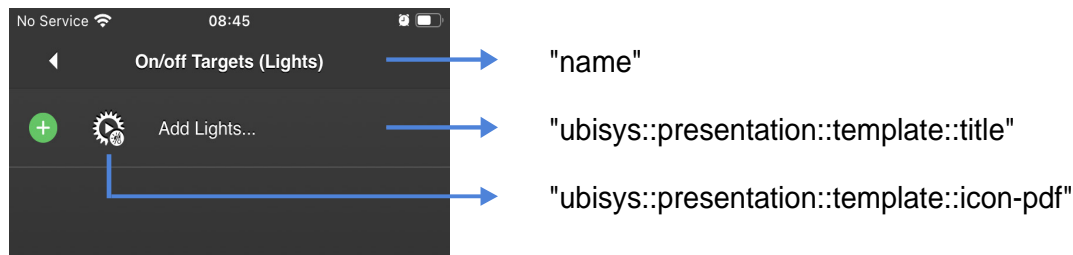


Figure 3. Template title and Icon

7.1.4.1. type

Mandatory

The type attribute represents the data type of the given element. Here in this case, "properties" is not confined to any specific data type like string, integer, etc. so it must be defined as an "object" to represent the generic nature of the data type.

7.1.5. properties

Mandatory

Properties is one of the mandatory elements of the configuration.schema which must include all possible configurable properties of the given automation template. The properties attribute must have the same overall structure as the JSON schema. Based on the automation template requirements, each of these properties must have type, name, default value, etc. attributes. These properties are displayed in the user interface with which the user can configure them based on the surrounding conditions in run time. Please note, the attributes must only be defined here. The actual logic and corresponding actions must be present in the logic file. Each property is based on a specific data type. As per the requirements of the automation template, the developer needs to decide the data type for all given properties and define it using corresponding attributes. The supported data types are as follows.

7.1.6. Data type and supporting attributes

7.1.6.1. type

This attribute is used to declare the data type of the element thereby defining how it will be processed. It can have one of the following values.

Table 2. Supported Data types

Data type	Description
string	For String type
object	For generic object
array	To create an array of elements.
integer	For integer data type
number	For float data type
boolean	For boolean data type

More details on how to use of these attributes are discussed in the [Section 7.2](#). Based on the data type, the developer can also define the range of valid values or define the default value by means of following attributes.

7.1.6.2. default

The default value of the given element must be provided via this attribute. It should follow the exact same data type as defined for the given element.

7.1.6.3. minimum

The minimum possible value that the given property is allowed to have is maintained by this attribute. It must only be used in case "type" is either "integer" or "number".

7.1.6.4. maximum

This attribute holds the maximum possible number that the given property can have. It must only be used in case "type" is defined as either "integer" or "number".

7.1.6.5. minItems

The minItems defines the number of minimum items necessary in the array. The ubisys application verifies if the minimum number of elements is there, with reference to the number mentioned by this attribute. The data type should be an integer. It must only be used in case "type" is defined as an "array".

7.1.6.6. maxItems

Same as in minItems, the maximum number of items necessary in the array is defined by maxItems. The data type should be an integer and overall rule **minItems** \leq **maxItems** must follow. It must only be used in case "type" is defined as an "array".

7.1.6.7. ubisys::type

Apart from the usual data types defined earlier, the developer can also use additional data types mentioned in this section. These special data types allows developers to incorporate unique functionality in their automation template. Each of these types have a specific significance as described below.

Table 3. Additional Data types

ubisys::type	Description
zigbee-application-instance	To define the application filter.
time-of-day	To represent time of a day in minutes format
color-temperature-k	To represent color temperature in Kelvin format

ubisys::type	Description
percentage	To use percentage format
cross-reference	To reference two different items
lightlevel:lux	To represent the illuminance value in the lux format

For more details on the use of these attributes, please refer to the [Section 7.2](#).

Once the data type, its default value, possible value range, etc is defined, the developer can describe more details of the given property by help of attributes defined in the following sub sections. Based on the following attributes, the user interface will be generated.

7.1.7. name

The main title of the section is defined by this attribute. The value must be in the form of a string. See [Figure 4](#) for more details.

```
1 "name": "Illuminance Sensors"
```

7.1.8. description

The description must contain a detailed explanation of the given element using the "string" data type. The user can go into more options in order to see this description. For example:

```
1 "description": "<p>The set of illuminance sensors available in the network. The illuminance sensor can provide light level information for the entire zone. You can also provide information on multiple sensors if required.</p>"
```

7.1.9. ubisys::short-description

The short-description appears just after the title of the given element. It must describe the element in a short "string". See [Figure 4](#) for more details.

```
1 "ubisys::short-description": "Set of illuminance sensors"
```

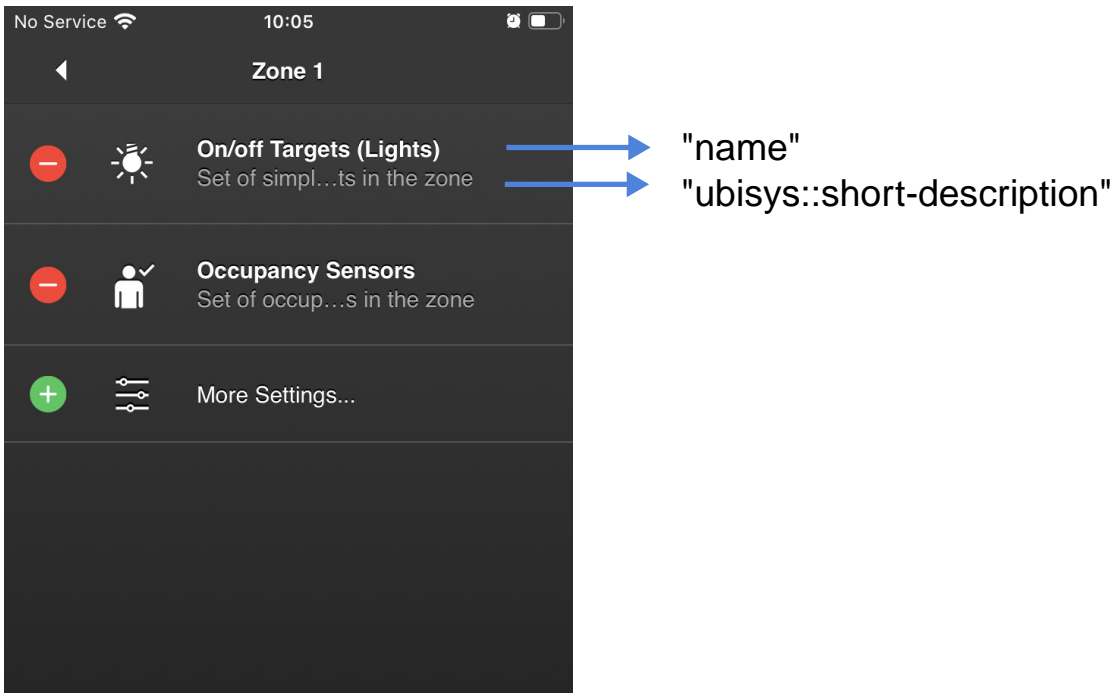


Figure 4. Automation template user interface

7.1.10. ubisys::presentation::order

The presentation order sequence will be considered while creating an index of the visual elements. Developers can use this attribute while creating a list of elements in order to define their specific sequence.

```
1 "ubisys::presentation::order": 0
```

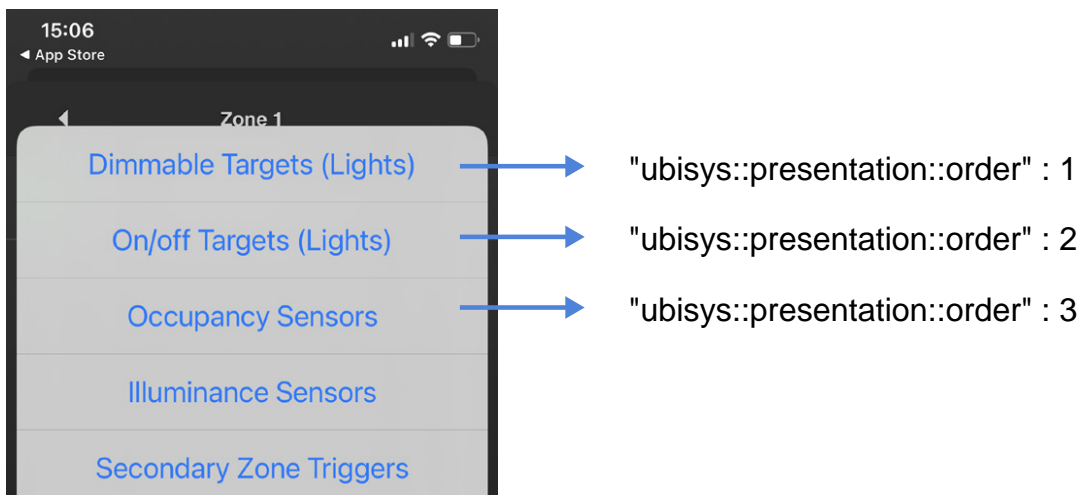


Figure 5. Presentation order sequence

7.1.11. ubisys::presentation::placeholder

The placeholder represents the position of the element in the user interface.

```
1 "ubisys::presentation::placeholder": "Zone"
```


7.1.12. ubisys::presentation::key

The presentation key defines if the element is supposed to be shown in the present user interface or it was shown in the parent interface. It is of type boolean and must have a value of either true or false.

```
1 "ubisys::presentation::key": true
```

7.1.13. ubisys::presentation::mandatory

The mandatory field is of type boolean and defines if the given element is optional or mandatory for a user to configure.

```
1 "ubisys::presentation::mandatory": false
```

7.1.14. ubisys::presentation::visible

Similar to the mandatory property, the visible property is of type boolean and defines if the visibility of the given element is hidden or visible from the start.

```
1 "ubisys::presentation::visible": true
```

7.1.15. ubisys::value::cross-reference

To reference two different items or properties of the zones this attribute can be used.

```
1 "ubisys::value::cross-reference" : "/zones"
```

7.1.16. ubisys::application-filter

With this attribute, the developer can define a filter to group together specific types of devices. Based on the filter definition, all devices available in the Zigbee network that qualify for this filter will be presented on the user interface. The user can then select the devices and add them to the automation template. Only these devices will then be considered in the context of the automation template and will behave based on the logic defined in the JavaScript file.

Example filter for "On Off Type Devices"

```
1 "ubisys::application-filter" :  
2 {  
3   "qualifying-device-types": [ { "profile-id": 260 },  
4                               { "profile-id": 49246 } ],  
5   "disqualifying-device-types": [],  
6   "qualifying-clusters" : [ { "id": 6, "client": false } ],  
7   "disqualifying-clusters" : [],  
8   "include-groups": true  
9 }
```

In this use-case, we will select "profile-id": 260 (home automation) and let server instances select

cluster "id": 6 (OnOff cluster).

Example filter for "Motion Sensor"

```
1 "ubisys::application-filter" :  
2 {  
3   "qualifying-device-types": [ { "profile-id": 260 },  
4                               { "profile-id": 49246 } ],  
5   "disqualifying-device-types": [],  
6   "qualifying-clusters" : [ { "id": 1030, "client": false } ],  
7   "disqualifying-clusters" : [],  
8   "include-groups": false  
9 }
```

The home automation devices are selected with "profile-id": 260 and occupancy sensors are filtered by selecting the server instance of the cluster "id": 1030 (Occupancy Sensing). The groups can be excluded from this filter with "include-groups": false.

As seen in the examples, there are four properties that need to be considered to define this filter.

7.1.16.1. qualifying-device-types

List of profile-ids and device-ids of devices that must qualify in this filter.

7.1.16.2. disqualifying-device-types

List of profile-ids and device-ids of devices that should not qualify in this filter.

7.1.16.3. qualifying-clusters

List of cluster-ids and cluster direction that must qualify in this filter.

7.1.16.4. include-groups

Boolean value to define if groups should be included or not. true (groups will be included) or false (groups will be excluded).



A profile identifier is 16 bits in length and specifies the application profile being used. A device identifier is 16 bits in length and specifies a specific device within Zigbee standards. A cluster identifier is 16 bits in length and identifies an instance of an implemented cluster specification. Please refer to Zigbee Cluster Library Specification for more details.

7.1.17. uniqueItems

This attribute must be used in order to define if the particular element is unique. The data type is boolean and must be used as follows

```
1 "uniqueItems": true
```

7.1.18. ubisys::value::full-scale

The full-scale value represents the possible full-scale number which is equivalent to the maximum possible value in most cases. The data type must be identical to that of the element.

```
1 "ubisys::value::full-scale": 254
```

7.1.19. ubisys::value::stepping

This represents the stepping value that will be used by the script to move from the initial value to the final value. The data type must be identical to that of the element.

```
1 "ubisys::value::stepping": 1
```

7.1.20. items

If the data type is "array" then the ubisys application will look into the "items" section of the element that describes these values. It must follow the minItems, maxItems boundaries and must use the same data type as declared before.

```
1 "daily_turn_off" :
2 {
3   "type": "array",
4   "minItems": 7,
5   "maxItems": 7,
6   "name": "Daily Turn Off",
7   "default": [ 79200, 79200, 79200, 79200, 79200, 79200, 79200 ],
8   "description": "<p>Determines the device turn off time.</p>",
9   "ubisys::short-description": "Daily times for turning off lights",
10  "items":
11  [
12    {
13      "type": "integer",
14      "name": "Sundays",
15      "description": "<p>The device turn off time for Sundays.</p>",
16      "ubisys::short-description": "Turn off time on Sundays",
17      "ubisys::presentation::icon-pdf": "#schedules.pdf",
18      "ubisys::type": "time-of-day"
19    },
20    {
21      "type": "integer",
22      "name": "Mondays",
23      "description": "<p>The device turn off time for Mondays.</p>",
24      "ubisys::short-description": "Turn off time on Mondays",
25      "ubisys::presentation::icon-pdf": "#schedules.pdf",
26      "ubisys::type": "time-of-day"
27    },
28    "... In total 7 items"
29  ]
30 }
```

7.1.21. ubisys::value::options

An option is a dictionary of different elements which is shown to the user to select from. The key must be a value of the data type defined by "type" and the values must be in a dictionary with a "name". This name will be shown to the user for option selection. For example:

```
1 "hold_time" :
2 {
3   "type": "integer",
4   "name": "Hold On Time",
5   "ubisys::type" : "timespan",
6   "minimum": 0,
7   "maximum": 86400,
8   "default": 600,
9   "ubisys::value::options" :
10  {
11    "00000": { "name": "none" },
12    "00005": { "name": "5 seconds" },
13    "00010": { "name": "10 seconds" },
14    "00015": { "name": "15 seconds" },
15    "00030": { "name": "30 seconds" },
16    "00060": { "name": "1 minute" },
17    "00120": { "name": "2 minutes" },
18    "00180": { "name": "3 minutes" },
19    "00240": { "name": "4 minutes" },
20    "00300": { "name": "5 minutes" },
21    "00360": { "name": "6 minutes" },
22    "00420": { "name": "7 minutes" },
23    "00480": { "name": "8 minutes" },
24    "00540": { "name": "9 minutes" },
25    "00600": { "name": "10 minutes" }
26  }
27 }
```

7.2. Examples

7.2.1. Properties examples using type

This chapter includes a variety of examples showcasing the use of different data types to define the property. These examples also represent the use of attributes as defined in previous sections.

7.2.1.1. String attribute

```
1 "zone_identifier":
2 {
3   "type": "string",
4   "name": "Zone",
5   "default": "Zone",
6   "description": "<p>This is the first zone.</p>",
7   "ubisys::short-description": "Definition of a default zone",
8 }
```

The given example shows the declaration of a string data type for a property named "zone_identifier" with a default value. For the UI purpose name, description, short-description can be added. As the

data type is string, no limiting attributes like minimum, maximum, minItems, maxItems are required.

7.2.1.2. Array attribute

```
1 "daily_turn_off" :
2 {
3   "type": "array",
4   "minItems": 7,
5   "maxItems": 7,
6   "name": "Daily Turn Off",
7   "default": [ 79200, 79200, 79200, 79200, 79200, 79200, 79200 ],
8   "description": "<p>Determines turn-off time of devices.</p>",
9   "ubisys::short-description": "Turn-off time of devices",
10  "items":
11  [
12    {
13      "type": "integer",
14      "name": "Sundays",
15      "description": "<p>Device Turn off time on Sundays.</p>",
16      "ubisys::short-description": "Turn off time on Sundays",
17      "ubisys::presentation::icon-pdf": "#schedules.pdf",
18      "ubisys::type": "time-of-day"
19    },
20    {
21      "type": "integer",
22      "name": "Mondays",
23      "description": "<p>Device Turn off time on Mondays.</p>",
24      "ubisys::short-description": "Turn off time on Mondays",
25      "ubisys::presentation::icon-pdf": "#schedules.pdf",
26      "ubisys::type": "time-of-day"
27    },
28    "... In total 7 items"
29  ]
30 }
```

The given example shows the declaration of an array data type for a property named "daily_turn_off" with a default value. As the data type is "array", it must have minItems, maxItems and the default must also be defined with the same size. The contents of the array are provided by the items attribute.

7.2.1.3. Integer attribute

```
1 "primary_level_value" :
2 {
3   "type": "integer",
4   "name": "Primary Level Value",
5   "default": 178
6   "minimum": 1,
7   "maximum": 254,
8   "description": "<p>Initial value of brightness</p>",
9   "ubisys::short-description": "Initial value"
10 }
```

The given example shows the declaration of an integer data type for a property named "primary_level_value" with a default value. Again, for UI purposes, a description can be added along with minimum/ maximum limits. The data type "integer" can have minimum, maximum limiting

attributes. If defined, they must be of the same data type (integer in this example).

7.2.1.4. Number attribute

```
1 "pi_proportional_gain" :
2 {
3   "type": "number",
4   "name": "Proportional Gain Multiplier",
5   "minimum": 0.05,
6   "maximum": 1,
7   "default": 0.3,
8   "description": "<p>The gain value for PI controller.</p>",
9   "ubisys::short-description": "Gain for PI controller",
10  "ubisys::value::options" :
11  {
12    "0.10": { "name": "0.1" },
13    "0.20": { "name": "0.2" },
14    "0.30": { "name": "0.3" },
15    "0.40": { "name": "0.4" },
16    "0.50": { "name": "0.5" },
17    "0.60": { "name": "0.6" },
18    "0.70": { "name": "0.7" },
19    "0.80": { "name": "0.8" },
20    "0.90": { "name": "0.9" },
21    "1.00": { "name": "1.0" }
22  }
23 }
```

The given example shows the declaration of a number data type which is equivalent to the float data type for a property named "pi_proportional_gain". Similar to "integer", the "number" can also have minimum, maximum in float format.

7.2.1.5. Boolean attribute

```
1 "enable_PI_controller" :
2 {
3   "type": "boolean",
4   "name": "Incorporate PI Controller",
5   "description": "<p>Whether to include the PI controller.</p>",
6   "ubisys::short-description": "Enable the PI controller",
7   "default": true
8 }
```

The boolean data allows developers to include a true/ false type of property. Such a property is forbidden to have minimum, maximum, minItems, maxItems attributes. The given example shows the declaration of a property named "enable_PI_controller".

7.2.1.6. Object attribute

```

1 "Further_items":
2 {
3   "type": "object",
4   "name": "Zone",
5   "description": "<p>Definitions for a single zone</p>",
6   "properties":
7   {
8     "... Additional properties"
9   }
10 }

```

The given example shows the declaration of an object data type for a property named "Additional_properties". For properties of the generic data type which can not be included in one of the options discussed before, "object" is available. Such properties cannot have minimum, maximum, minItems, maxItems attributes.

7.2.2. Properties examples using ubisys::type

Following are some of the examples on how developers can use special data types mentioned in [Section 7.1.6.7](#) to define the properties.

7.2.2.1. zigbee-application-instance attribute

```

1 "items":
2 {
3   "type": "string",
4   "ubisys::type": "zigbee-application-instance",
5   "ubisys::application-filter" :
6   {
7     "qualifying-device-types": [],
8     "disqualifying-device-types": [],
9     "qualifying-clusters" : [],
10    "disqualifying-clusters" : [],
11    "include-groups": false
12  }
13 }

```

The data type must be string and the ubisys::type must be zigbee-application-instance. The developer must use this data type to define the application-filter by means of device types, cluster ids as discussed in [Section 7.1.16](#).

7.2.2.2. time-of-day attribute

```

1 "start_time" :
2 {
3   "type": "integer",
4   "name": "Start Time",
5   "ubisys::type": "time-of-day",
6   "description": "<p>Activation time of the automation.</p>",
7   "default": 21600
8 }

```

This type is designated to represent a specific time of the day in the integer data type, where the number represents time in minutes. So 10:00 AM is represented as 36000 (i.e. 10 * 3600). The property should have a name and default value as shown in the above example.

7.2.2.3. color-temperature-k attribute

```
1 "initial_color_temp_k" :
2 {
3   "type": "integer",
4   "name": "Initial Color Temperature Value",
5   "ubisys::type": "color-temperature-k",
6   "minimum": 1800,
7   "maximum": 7500,
8   "default": 5000
9 }
```

The value of the color temperature can be represented by units of kelvin, using the "color-temperature-k" ubisys::type. The data type must be integer and it can also have name, a default value as shown.

7.2.2.4. percentage attribute

```
1 "initial_color_temp_k" :
2 {
3   "type": "integer",
4   "name": "Initial Brightness level",
5   "ubisys::type": "percentage",
6   "ubisys::value::full-scale": 120,
7   "default": 70
8 }
```

Percentage being a widely used term in automation scripts, a separate type has to be dedicated for it. Developers can use it to define a property as seen in the example. The data type must be integer.

7.2.2.5. cross-reference attribute

```
1 "items" :
2 {
3   "type": "string",
4   "ubisys::type": "cross-reference"
5   "ubisys::value::cross-reference" : "/zones"
6 }
```

This type can be used to reference two different zones in case they are adjacent. The data type must be string.

7.2.2.6. lightlevel:lux attribute


```
1 "target_light_level" :  
2 {  
3   "type": "number",  
4   "name": "Target Light Level",  
5   "ubisys::type" : "lightlevel:lux",  
6   "minimum": 0,  
7   "maximum": 100000,  
8   "default": null  
9 }
```

"lux" is the unit used to measure the intensity of the light level, hence a dedicated `ubisys::type` is used to annotate this unit. The data type must be a "number".

7.2.3. "Motion-based lighting control system" use case

Following is an example of configuration.schema.json for a use case of a motion-based lighting control system. Please note, that the following schema example will only generate the required user interface. In order to get the automation running, the respective logic JavaScript file, as well as other mandatory bundle files, need to be present.

```
1 {
2   "$schema": "http://json-schema.org/schema#",
3   "$id": "http://your.organisation.com/configuration.json",
4   "type": "object",
5   "ubisys::presentation::template::title": "More Settings...",
6   "ubisys::presentation::template::icon-pdf": "#settings.pdf",
7   "properties":
8   {
9     "motion_sensors":
10    { "..."},
11    "onoff_devices":
12    { "..."},
13    "enable_automation" :
14    { "..."}
15  }
16 }
```

Inside "properties", there are three sub-items, out of which the first property is the motion_sensors, which is a filter to group a certain type of devices. The name and description are updated and an icon is selected to represent the use of this property with presentation::order set to 1. The type is an "array" with elements present in the "items" section. To filter out devices, "ubisys::type" is set to "zigbee-application-instance" and the application filter is configured to select motion-sensing devices.

```

1 "motion_sensors":
2 {
3   "type": "array",
4   "name": "Motion Sensors",
5   "description": "<p>Motion sensors in the zone.</p>",
6   "ubisys::short-description": "Motion sensors",
7   "ubisys::presentation::icon-pdf" : "#occupancy.pdf",
8   "ubisys::presentation::template::title": "Add Sensors...",
9   "ubisys::presentation::template::icon-pdf": "#occupancy.pdf",
10  "ubisys::presentation::mandatory": true,
11  "ubisys::presentation::visible": true,
12  "ubisys::presentation::order": 1,
13  "items":
14  {
15    "type": "string",
16    "ubisys::type": "zigbee-application-instance",
17    "ubisys::application-filter" :
18    {
19      "qualifying-device-types": [ { "profile-id": 260 },
20                                { "profile-id": 49246 } ],
21      "disqualifying-device-types": [],
22      "qualifying-clusters" : [ { "id": 1030,
23                                "client": false } ],
24      "disqualifying-clusters" : [],
25      "include-groups": false
26    }
27  }
28 }

```

Similarly the onoff_devices property is configured to select devices with the onoff cluster included.

```

1 "onoff_devices":
2 {
3   "type": "array",
4   "name": "On/off light devices",
5   "description": "<p>The set of on/off lights in the zone.</p>",
6   "ubisys::short-description": "Set of on/off lights",
7   "ubisys::presentation::icon-pdf" : "#application-lighting.pdf",
8   "ubisys::presentation::template::title": "Add Light Devices...",
9   "ubisys::presentation::title": "Select On/off Light Devices",
10  "ubisys::presentation::template::icon-pdf": "bundle.pdf",
11  "ubisys::presentation::mandatory": true,
12  "ubisys::presentation::visible": true,
13  "ubisys::presentation::order": 2,
14  "items":
15  {
16    "type": "string",
17    "ubisys::type": "zigbee-application-instance",
18    "ubisys::application-filter" :
19    {
20      "qualifying-device-types": [ { "profile-id": 260 },
21                                  { "profile-id": 49246 } ],
22      "disqualifying-device-types": [],
23      "qualifying-clusters" : [ { "id": 6, "client": false } ],
24      "disqualifying-clusters" : [],
25      "include-groups": true
26    }
27  }
28 }

```

The third property is of type boolean with the name `enable_automation`, to decide if the automation logic should be enabled or disabled.

```

1 "enable_automation" :
2 {
3   "type": "boolean",
4   "name": "Incorporate PI Controller",
5   "description": "<p>Whether to include the PI controller.</p>",
6   "ubisys::short-description": "Enable the PI controller",
7   "default": true,
8   "ubisys::value::options" :
9   {
10    "true": { "name": "Enable automation template" },
11    "false": { "name": "Disable automation template" }
12  }
13 }

```

An example of a JavaScript file implementing the automation logic by utilizing the above-mentioned properties is available as shown below.

```

1 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2 // Global variables
3
4 // For Zigbee apis
5 var Zigbee = require('sys/zigbee');
6
7 // To support timer functionalities
8 var Timer = require('sys/timers');
9
10 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
11 // The configuration is supplied to this script by the ubisys Zigbee
12 // JavaScript Engine. Its contents will be created by the end user using the
13 // ubisys the mobile application. These configurations were defined by the //
14 // configuration.schema.json file
15
16 var config = instanceConfiguration();
17
18 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
19 // Functions
20
21 // Called whenever the occupancy attribute changes
22 function onOccupancyStateChange(attributeID, attribute)
23 {
24     print('motion sensor value changed to', attribute);
25
26     if (config.enable_automation)
27     {
28         if (attribute.testBit(1))
29         {
30             // Turn on the Lights
31             config.onoff_devices.getOnOffCluster().on(function(status)
32             {
33                 print('Lights turned on:', status)
34             });
35         }
36         else
37         {
38             // Turn off the lights
39             config.onoff_devices.getOnOffCluster().off(function(status)
40             {
41                 print('Lights turned off:', status)
42             });
43         }
44     }
45     else
46     {
47         print('Automation template is disabled.')
48     }
49 }
50
51 // Main function
52 Zigbee.onReady(function()
53 {
54     // Get the occupancy sensing cluster
55     var occupancy = config.motion_sensors.getApplication(1).getCluster(0x0406);
56
57     // Register for attribute changes on attribute 0 (Occupancy, bitmap)
58     occupancy.onAttributeChanged(0x0000, onOccupancyStateChange);
59 });

```

The Zigbee variable allows the use of all Zigbee functionality, with which connected devices can be managed. The timer variable provides basic support for timer operations, which can be useful to create time-based events. Once the Zigbee.onReady is called, the automation initialization will start, which will use the Zigbee functionality to register an event for an occupancy attribute change. Thus, any changes in the occupancy bit triggers the onOccupancyStateChange() function call. Inside this function, it checks the value enable_automation property and controls the onoff_devices.

8. Adding support for other languages

The bundle-info.json and configuration-schema must be in the English language. But there can be situations where a developer wants to add support for other languages. So this chapter explores a step-by-step procedure on how the developer can add multilingual automation templates. While loading the automation template, the ubisys application first loads the default files and generates the localized version of the configurations. Then the application checks the system language and if it is not English, it loads the corresponding language file from the automation template bundle and updates the localized configurations version. Once done, this will be used to create the user interface.



This is an optional feature and it is up to the developer to add support for additional languages.

To add support for a new language, the developer needs to create a patch file that includes instructions and texts in the new language. With the current state, it is mandatory to provide two files for each newly supported language, namely configuration.XX.patch.schema.json and bundle-info.XX.patch.json. Here XX stands for language code in "ISO 639-1" code. So to support the German (Deutsch) language, the filenames must be bundle-info.de.patch.json and configuration.de.patch.schema.json.

These files should have an array of JSON containing operation, path and value. The ubisys application supports following operations

Table 4. Supported operations

Operation	Description
"add"	To append the contents
"remove"	To remove the contents
"replace"	To replace the contents with the given "value"
"move"	To remove the contents from "from" location and add it to "path" location
"copy"	To copy the contents from "from" location and add it to "path" location
"test"	To check if the value matches the "value" present at "path"

Example to update bundle-info.json

```
1 {
2   "id": "lighting.example",
3   "category": "lighting",
4   "keywords": [ "illuminance", "sensors", "HCL"],
5   "version": "1.0.0",
6   "publisher": "Organisation name",
7   "publisher-id": "organisation.id",
8   "name": "Lighting Control Script",
9   "description": "Lighting control script for home automation",
10  "bundle-html": "bundle_information.html",
11  "item-description-html": "item_description.html",
12  "bundle-pdf": "bundle_image_icon.pdf",
13  "logic": "automation_script.js",
14  "configuration-schema": "configuration.schema.json"
15 }
```

bundle-info.de.patch.json must look like

```

1 [
2   { "op": "replace",
3     "path": "/name",
4     "value": "name_in_other_language"
5   },
6   { "op": "add",
7     "path": "/description",
8     "value": " and systems"
9   }
10 ]

```

The resultant localized JSON would look like

```

1 {
2   "id": "lighting.example",
3   "category": "lighting",
4   "keywords": [ "illuminance", "sensors", "HCL"],
5   "version": "1.0.0",
6   "publisher": "Organisation name",
7   "publisher-id": "organisation.id",
8   "name": "name_in_other_language",
9   "description": "Lighting control script for home automation and systems",
10  "bundle-html": "bundle_information.html",
11  "item-description-html": "item_description.html",
12  "bundle-pdf": "bundle_image_icon.pdf",
13  "logic": "automation_script.js",
14  "configuration-schema": "configuration.schema.json"
15 }

```

As seen, there are two operations, one to replace the contents present on the path "/name" with the given value "name_in_other_language". So the application will look for the path into the bundle-info.json and replace its contents. And the second operation is to add contents. Similarly, the configuration.de.patch.schema.json can be generated to update the content of configuration.schema.json to update the user interface. As such, the developer can add any number of operations based on the template requirements.

9. index.plist

The plist is an important file that provides details on the available automation bundle files on the webserver. For every newly created automation bundle, the developer should add an entry in the index.plist file. The basic structure with one entry of an automation bundle is shown here

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
  "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
3 <plist version="1.0">
4 <array>
5   <dict>
6     <key>id</key>
7     <string>unique_id_of_automation_script</string>
8     <key>icon-pdf</key>
9     <string>icon_of_the_automation_script</string>
10    <key>en</key>
11    <dict>
12      <key>name</key>
13      <string>name_of_the_automation_script</string>
14      <key>description</key>
15      <string>short_description_of_the_automation_script</string>
16    </dict>
17  </dict>
18 </array>
19 </plist>
```

The basic skeleton of the document includes an array of dictionaries, where each dictionary belongs to a unique automation bundle file present on the webserver. Each dictionary should have three key-value pairs.

id The unique id of the automation bundle is provided in string format. The contents must match with the id from the bundle-info.json file.

icon-pdf The icon pdf file will be used to display an icon in front of the automation bundle name in the ubisys application.

en The third entry is another dictionary that contains the name and a short description of the automation bundle file. The developer must provide these details based on the contents of the automation bundle files.

Once the index.plist is parsed, the provided array will be mapped into the user interface from where the user can select a specific automation bundle. Based on the user selection, the bundle file will be downloaded and processed further.

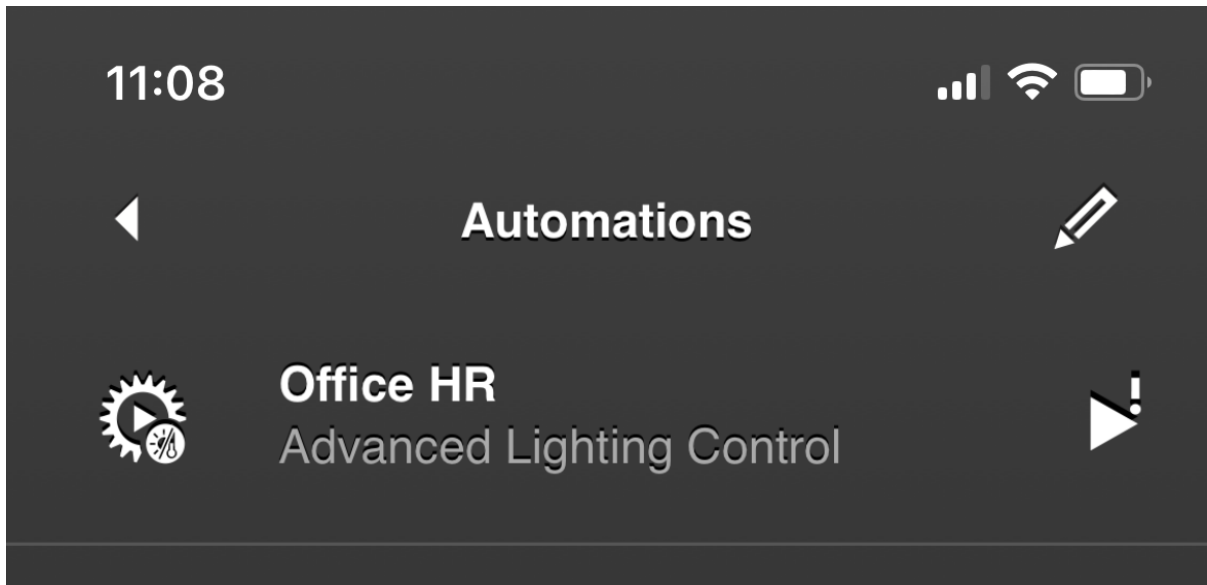
10. Parsed JSON file

Once the bundle file is parsed by the ubisys application, it will generate a JSON file that summarizes the configuration schema. This file includes all the properties and their respective values (the ones selected by the user or default ones) and will be forwarded to the ubisys gateway for further processing. The configuration variables in automation JavaScript file will be initialised with these values and will be used accordingly. Following is a snippet from one of such a file generated by the ubisys application. Please note, the developer is not required to create or configure such a file, this example is just to give some insights into the functional behavior of the ubisys application.

```
1 {
2   "configuration":
3   {
4     "daily_off": [82800, 0, 0, 0, 0, 0, 82800],
5     "daily_operating_times":
6     [
7       [21600, 79200],
8       [18000, 82800],
9       [18000, 82800],
10      [18000, 82800],
11      [18000, 82800],
12      [18000, 82800],
13      [21600, 79200]
14    ],
15     "hold_time_on": 900,
16   },
17   "description": "Bedroom configuration",
18   "enabled": 1,
19   "template": "your.template.settings",
20   "uuid": "1234-1234-0000-0000-12345678"
21 }
```

11. Troubleshooting_errors

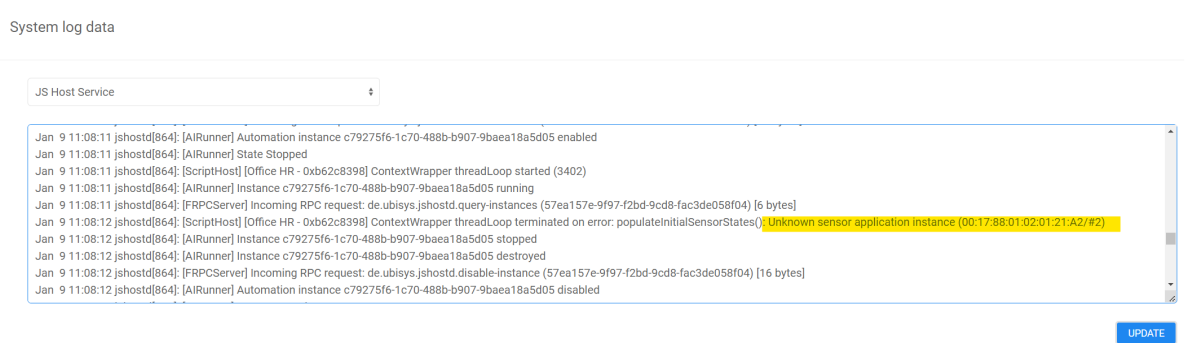
In case of the automation stopping you will see a greyed out start button with an exclamation mark.



To find out what the cause is you can go to the web surface of the gateway, specifically "maintenance", which will display any errors in the system log data window. Here are two examples what the log window might show as causes:

a) Missing illuminance sensor, e.g. initially included in the automation but at some later point having been deleted from the inventory.

Solution - Delete the sensor from the automation template as well.



b) Missing target light, e.g. initially included in the automation but at some later point having been deleted from the inventory.

Solution - Delete the target light from the automation template as well.

System log data

JS Host Service



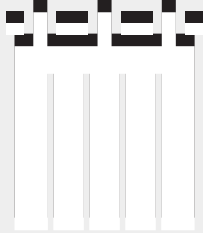

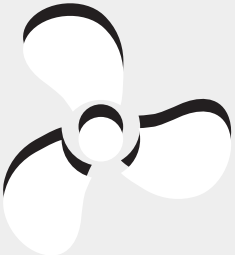
```
Jan 9 11:12:58 jshostd[864]: [AIRunner] Instance c79275f6-1c70-488b-b907-9baea18a5d05 running
Jan 9 11:12:58 jshostd[864]: [ScriptHost] [Office HR - 0xb62c8398] ContextWrapper threadLoop started (3547)
Jan 9 11:12:58 jshostd[864]: [FRPCServer] Incoming RPC request: de.ubisys.jshostd.query-instances (57ea157e-9f97-f2bd-9cd8-fac3de058f04) [6 bytes]
Jan 9 11:12:59 jshostd[864]: [ScriptHost] [Office HR - 0xb62c8398] ContextWrapper threadLoop terminated on error: populateTargetDevices(): Unknown device application instance (00:1F:EE:00:00:SE:D7/#1)
Jan 9 11:12:59 jshostd[864]: [AIRunner] Instance c79275f6-1c70-488b-b907-9baea18a5d05 stopped
Jan 9 11:12:59 jshostd[864]: [AIRunner] Instance c79275f6-1c70-488b-b907-9baea18a5d05 destroyed
Jan 9 11:12:59 jshostd[864]: [FRPCServer] Incoming RPC request: de.ubisys.jshostd.query-templates (57ea157e-9f97-f2bd-9cd8-fac3de058f04) [6 bytes]
Jan 9 11:12:59 jshostd[864]: [FRPCServer] Incoming RPC request: de.ubisys.jshostd.query-instances (57ea157e-9f97-f2bd-9cd8-fac3de058f04) [6 bytes]
Jan 9 11:12:59 jshostd[864]: [FRPCServer] Incoming RPC request: de.ubisys.jshostd.get-instance-configuration (57ea157e-9f97-f2bd-9cd8-fac3de058f04) [16 bytes]
```

UPDATE

12. Integrated icons

The ubisys app supports some predefined icons, which developers can include in their customized scripts. In order to include them, "#" prefix (ex: "#application-lighting.pdf") must be used.

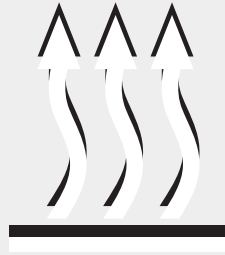
Table 5. Images supported by ubisys application

File name	Icon
actions.pdf	
application-awning.pdf	
application-curtain.pdf	
application-door-lock.pdf	
application-fan.pdf	

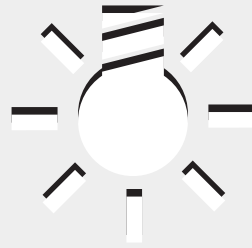
File name

Icon

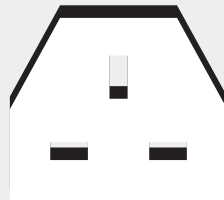
application-floor-heating.pdf



application-lighting.pdf



application-mains-outlet-uk.pdf



application-mains-outlet.pdf



application-other.pdf



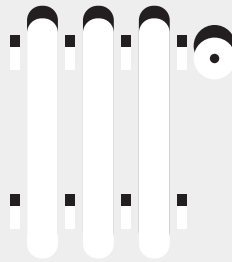
File name

Icon

application-projector-screen.pdf



application-radiator-heating.pdf



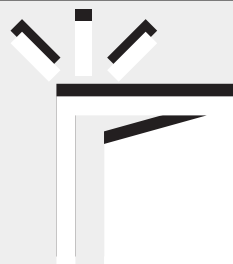
application-roller-exterior.pdf



application-roller.pdf



application-security-contact.pdf



File name

Icon

application-security-fire.pdf



application-security-keyfob.pdf



application-security-keypad.pdf



application-security-leakage.pdf



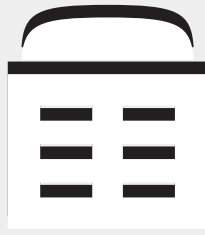
application-security-motion.pdf



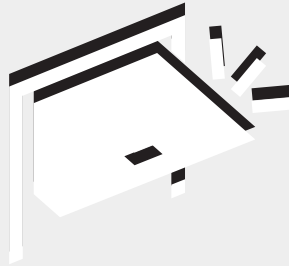
File name

Icon

application-security-siren.pdf



application-security-tilt.pdf



application-security-vibration.pdf



application-shade.pdf



application-shutter.pdf



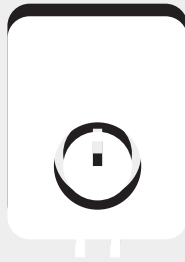
File name

Icon

application-unused.pdf



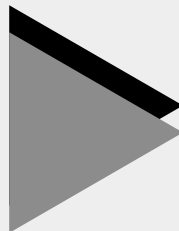
application-water-heater.pdf



application-window.pdf



automation-disabled.pdf



automation-failed.pdf



File name

Icon

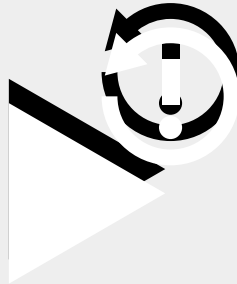
automation-new.pdf



automation-notinstalled.pdf



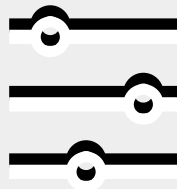
automation-restarting.pdf



automation-running.pdf



automation-settings.pdf



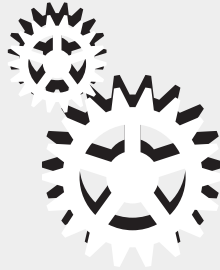
File name

Icon

automations.pdf



basic-configuration.pdf



battery-level-0.pdf



battery-level-100.pdf



battery-level-25.pdf



File name

Icon

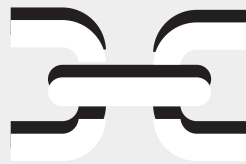
battery-level-50.pdf



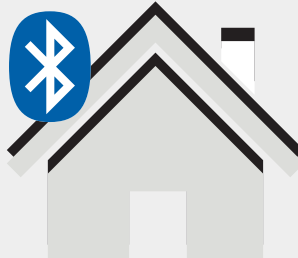
battery-level-75.pdf



bindings.pdf



blendz-gateway.pdf



button-back.pdf



File name

Icon

button-delete.pdf



button-discard.pdf



button-edit.pdf



button-next.pdf



button-save.pdf



File name

Icon

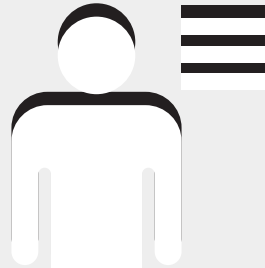
color.pdf



components-zigbee.pdf



configuration.pdf



control-action-off-on.pdf



control-action-on-off.pdf

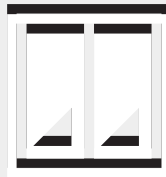


File name**Icon**

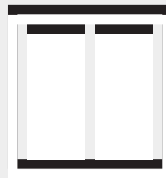
control-action-toggle.pdf



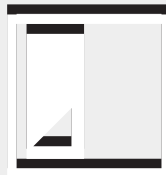
control-double-pushbutton.pdf



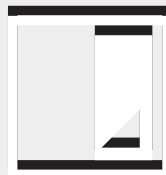
control-double-switch.pdf



control-pushbutton-left.pdf



control-pushbutton-right.pdf

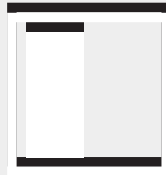


File name**Icon**

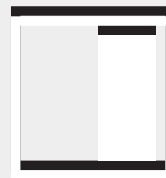
control-pushbutton.pdf



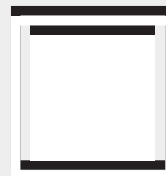
control-switch-left.pdf



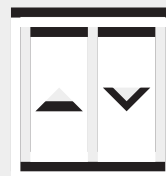
control-switch-right.pdf



control-switch.pdf



control-updown-switch.pdf



File name

Icon

ct-cool-white.pdf



ct-warm-white.pdf



dateAndTime.pdf



disabled.pdf



discard.pdf



File name**Icon**

door-lock-lock.pdf



door-lock-state-intermediate.pdf



door-lock-state-invalid.pdf



door-lock-state-locked.pdf



door-lock-state-unlocked.pdf



File name

Icon

door-lock-unlock.pdf



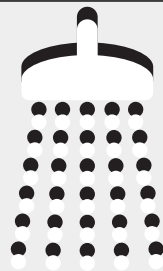
down.pdf



enabled.pdf



environment-bathroom.pdf



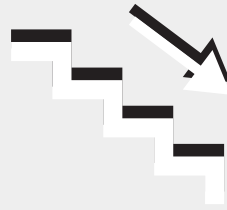
environment-bedroom.pdf



File name

Icon

environment-cellar.pdf



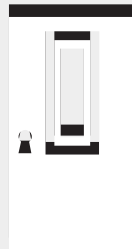
environment-diningroom.pdf



environment-dressingroom.pdf



environment-entry.pdf



environment-garage.pdf



File name

Icon

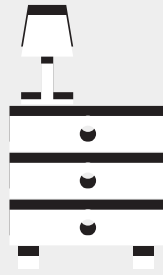
environment-garden.pdf



environment-guestroom.pdf



environment-hallway.pdf



environment-hobbyroom.pdf



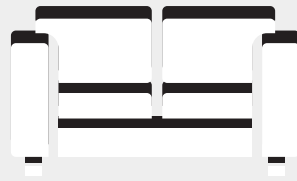
environment-kitchen.pdf



File name

Icon

environment-livingroom.pdf



environment-nursery.pdf



environment-office.pdf



environment-other.pdf



environment-pantry.pdf



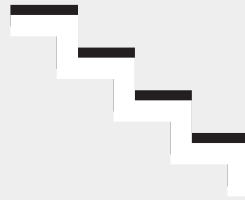
File name

Icon

environment-pool.pdf



environment-stairway.pdf



environment-storage.pdf



environment-watercloset.pdf



failure.pdf



File name

Icon

favorites.pdf



gateway.pdf



groups.pdf



home.pdf



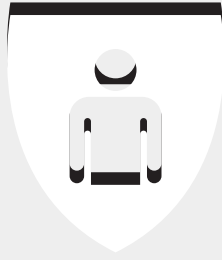
ias-arm-away.pdf



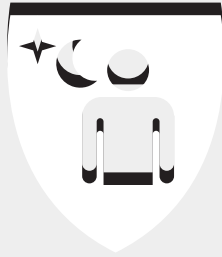
File name

Icon

ias-arm-day.pdf



ias-arm-night.pdf



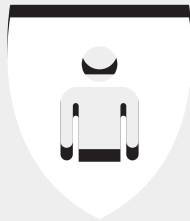
ias-armed-away-small.pdf



ias-armed-away.pdf



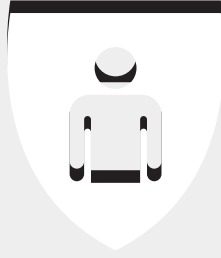
ias-armed-day-small.pdf



File name

Icon

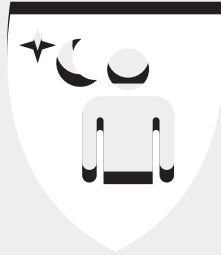
ias-armed-day.pdf



ias-armed-night-small.pdf



ias-armed-night.pdf



ias-arming-away.pdf



ias-arming-day.pdf



File name

Icon

ias-arming-night.pdf



ias-cie.pdf



ias-disarm.pdf



ias-disarmed.pdf



ias-entry-delay.pdf



File name

Icon

ias-exit-delay.pdf



ias-in-alarm-small.pdf



ias-in-alarm.pdf



ias-not-ready-to-arm-small.pdf



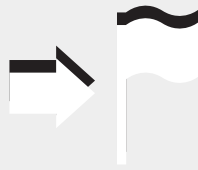
ias-not-ready-to-arm.pdf



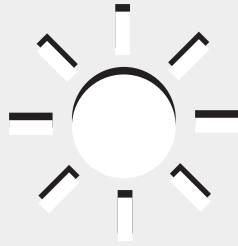
File name

Icon

identify.pdf



illuminance.pdf



interval.pdf



level.pdf



lift-maximum.pdf



File name

Icon

lift-minimum.pdf



locked.pdf



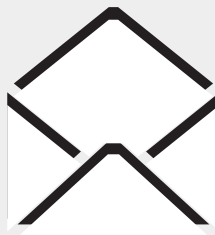
metering.pdf



network-discovery.pdf



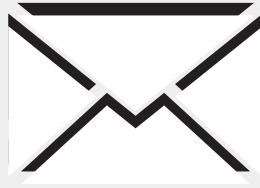
notification-viewed.pdf



File name

Icon

notification.pdf



occupancy.pdf



occupied.pdf



off.pdf



on.pdf



File name

Icon

price.pdf



relative-humidity.pdf



repair.pdf



scenes.pdf



schedules.pdf



File name**Icon**

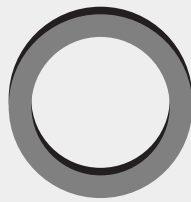
security-zone-all-clear.pdf



security-zone-battery-low.pdf



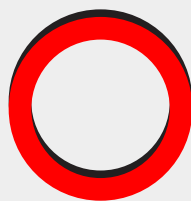
security-zone-invalid.pdf



security-zone-tampered.pdf



security-zone-warning.pdf



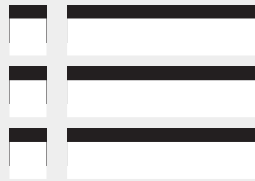
File name

Icon

security-zone-warning2.pdf



settings.pdf



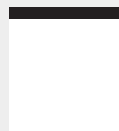
shade-lift.pdf



shade-tilt.pdf



stop.pdf



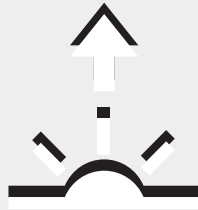
File name

Icon

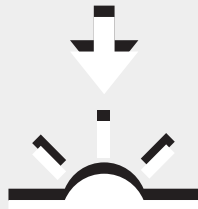
success.pdf



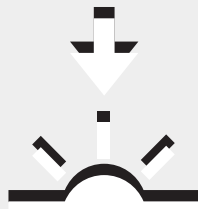
sunrise.pdf



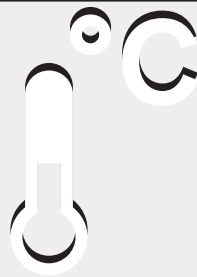
sunset.pdf



surveillance.pdf



temperature.pdf



File name

Icon

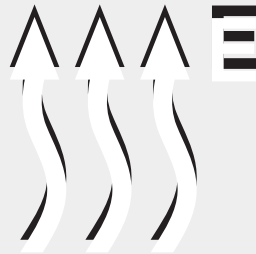
thermostat-automatic.pdf



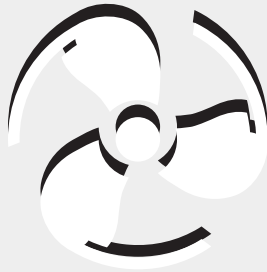
thermostat-cool.pdf



thermostat-emergency-heating.pdf



thermostat-fan-1.pdf



thermostat-fan-2.pdf



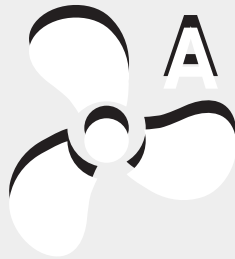
File name

Icon

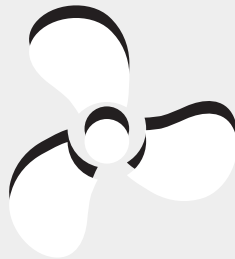
thermostat-fan-3.pdf



thermostat-fan-auto.pdf



thermostat-fan.pdf



thermostat-heat.pdf



thermostat-louver-0.pdf



File name

Icon

thermostat-louver-100.pdf



thermostat-louver-25.pdf



thermostat-louver-50.pdf



thermostat-louver-75.pdf



thermostat-louver-autoswing.pdf



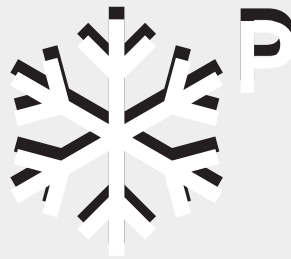
File name

Icon

thermostat-off.pdf



thermostat-precooling.pdf



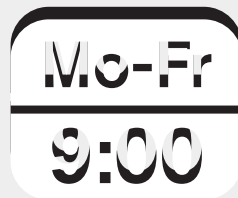
tilt-maximum.pdf



tilt-minimum.pdf



time.pdf



File name

Icon

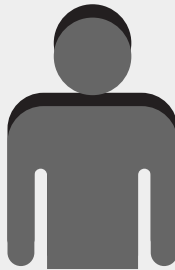
ubisys-g1.pdf



unlocked.pdf



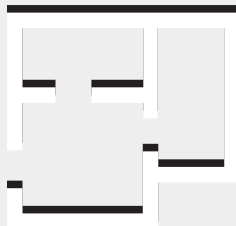
unoccupied.pdf



up.pdf



zones.pdf



II: Javascript

13. Features

The current feature set of the preview release contains the following features:

- Basic script editing and control, viewing of log messages
- Timers
 - Single-shot and periodic timers
 - Day timers and solar timers (at or relative to sunrise/sunset)
- Zigbee Integration
 - Generic support for all clusters and cluster-directed commands
 - Specialized support for the On/Off, Level Control and Window Covering clusters
 - Support for a subset of attribute types

14. Fundamentals

A basic user interface to edit and control scripts is provided and accessible via a web browser. Any number of scripts can be defined, after which each script can be enabled or disabled separately. If enabled, it will be executed until it is either manually disabled or stopped due to an error. All enabled scripts are run simultaneously and independently off each other. The execution of a script is event-based. This means that code contained in a script is executed in response to a certain event occurring, after which the script execution must be returned to the script runtime. The script must not (and cannot) block and wait for an event to occur (e.g. in a loop). When a script is started, it is loaded into memory and compiled to an internal representation. It is then executed once, i.e. any code in the main program (which is not encapsulated into functions) is executed once. During this initial execution, the script needs to register for any events which it desires to handle. Events could be, for example:

- a change of a Zigbee attribute on a device
 - a light switched on or off
 - brightness of a light changed
 - occupancy sensor state changed
- a timer
 - after a certain interval
 - triggered on a specific day time
 - triggered periodically
 - sunset/sunrise
- completion of a previously requested operation
 - a Zigbee command sent to a device Each script maintains a dedicated, cyclic log buffer into which log data is collected. The log data is accessible to the user/developer through the web browser.
Log data can either originate from the script runtime and its APIs itself or can be generated by the script via `print()` or `alert()` statements.

14.1. Naming conventions

JavaScript is a prototype-based language and does itself not know about classes or interfaces, but both terms are used in this document to describe the following concepts:

14.1.1. Class

A class refers to a public, accessibly named constructor function of the class name. All class member functions are publicly accessible functions, which are exported as properties of the created object instance.

14.1.2. Interface

An interface is similar to a class in that it exports publicly accessible functions, but does not have a user-visible/accessibly constructor function. It is therefore not possible to directly instantiate interfaces. An instance of a given interface is returned by various functions of the public API.

14.1.3. Inheritance

Contrary to class-based languages, inheritance in JavaScript is implemented via prototype chains. Class- and interface-based inheritance can easily be modelled via prototype chains. A class is said to inherit from another class (the base class) if it provides a superset of the base class' functions. Examples are the various clusters. The generic cluster implements functions to obtain attributes and send generic commands. Specialized derivatives exist for specific clusters: the OnOffCluster supports on(), off() and toggle() functions to act on the switch output and a function to monitor the on/off state. Besides that, it inherits the basic functionality of the generic cluster, e.g. getAttribute().

14.2. Firmware version references

Features introduced in newer firmware releases are indicated by the firmware version stated in rectangular brackets on the right of the feature, e.g. [1.5].

14.3. APIs/Interfaces

Several Application Programming Interfaces (APIs) are provided by the runtime to be utilized by scripts. All APIs are organized into so-called modules. A script must import the modules which it requires to be able to use them. Modules are imported via the require() function:

```
1 var Timer = require('sys/timers');
```

The module identification (sys/timer) is set by the runtime and identifies the module to be imported. The variable name (Timer) is set by the user/developer of the script and can be chosen freely, as permitted by the JavaScript language. Each module exports a certain number of functions, as specified in the module's documentation. They are available as properties on the variable assigned by the require() function:

```
1 var Timer = require('sys/timers');
2
3 Timer.createPeriodicTimer("5s", function()
4 {
5     // code will be executed every 5 seconds
6 });
```

14.4. Zigbee and Zigbee Cluster Library (ZCL) Basics

The Zigbee device model is defined by devices, applications, clusters and attributes. A device usually represents a physical Zigbee device, e.g. a dimmable light. Devices are primarily identified by their unique 64-bit IEEE address. Each device can (and usually does) host multiple applications. Applications are identified by their endpoint, within the range of 1 to 240. An application is a separate functional entity, encapsulation one function of the device. A dimmable light might consist of several applications: the actual light (switchable/dimmable output) plus one or several switch inputs. Each of these functions is represented by one application. Each application itself consists of various clusters. A cluster represents a standardized sub-unit of the application's function. A cluster is identified by its 16 bit cluster ID. Examples for clusters would be an On/Off cluster to switch an output, a Level Control to e.g. set the brightness of a dimmable lamp or a Color Control cluster to set the colour of a lamp. The Zigbee Cluster Library defines a pre-defined set of applications and clusters and states

mandatory and optional clusters for each application. Each cluster can receive (or generate) a certain set of commands, e.g. an “on” command to switch a light on (directed to the On/Off cluster). Besides that, each cluster defines a certain set of attributes, which represent the current state (e.g. On/Off state, brightness level, power consumption) or configuration of the cluster. Attributes are defined by their 16 bit attribute Id and can have various types, e.g. bool, unsigned/signed integers of various lengths etc.

15. Programming Interface

15.1. Globals and Builtins

Several free functions are provided in the global context:

- `require()` to import modules
- `print()` and `alert()` provide a way to log diagnostic output of a script. In a future revision, a dedicated logging API will be provided. Both functions may take multiple arguments, which will be separated by spaces. Standard string coercion rules apply, e.g. an Object's `toString()` method will be called, if available.

The runtime environment supports the standard ECMAScript builtins, e.g.:

- `Date`
- `Regex`
- `Error`, `RangeError`, `TypeError`, `SyntaxError`, `ReferenceError` constructors
- Various Buffer objects, please refer to [Section 15.4](#) for details.

15.2. Timer API

```
1 var Timer = require('sys/timers');
```

Allows the creation of timer events. Supports the following types of timers:

- Single-shot (e.g. in 10 seconds)
- Periodic (e.g. every 5 minutes)
- Fixed day time (at 10:30am every day)
- At or relative to sunrise or sunset

15.2.1. Exported functions

15.2.1.1. `createTimer()`

```
1 function createTimer(timespec, callback)
```

Parameters:

- timespec** specifies when the timer should fire (see below)
- callback** the function to invoke at the given time.

Returns:

The Timer instance (see [Section 15.2.2.1](#)). Ignore if not used.

Throws:

SyntaxError the timespec could not be parsed

RangeError a negative timeout was given

TypeError the callback is not callable

Format of the timespec parameter:

- a plain integral number indicates the time in seconds
- A combination of integral numbers with time suffixes (h/m[in]/s[ec]/ms)
- If multiple suffixes are used, they must be specified in the natural order (i.e. hours before minutes before seconds)

Examples of valid timespecs:

- 10 (10 seconds)
- 1h
- 1h30min

15.2.1.2. createPeriodicTimer()

```
1 function createPeriodicTimer(timespec, callback)
```

Same as createTimer(), except that the timers fires periodically.

15.2.1.3. createDayTimer()

```
1 function createDayTimeTimer(timespec, callback)
```

Parameters:

timespec specifies when the timer should fire (see below)

callback the function to invoke at the given time.

Returns:

The Timer instance (see [Section 15.2.2.1](#)). Ignore if not used.

Throws:

SyntaxError the timespec could not be parsed

RangeError a negative timeout was given

TypeError the callback is not callable

Format of the timespec parameter:

- hh:mm:ss
The day time (in 24-hour notion) when the timer should fire. The seconds part (:ss) is optional and may be left out.
- sunrise, sunrise+/-offset, sunset, sunset+/-offset A timer at or relative to the sunrise/sunset. Requires that the geographic position of the gateway is set.

Offset may be any of the following:

- a plain integral number representing seconds
- A combination of integral numbers with time suffixes (h/m[in]/s[ec]/ms)

Examples of valid timespecs:

- sunrise+1h
- sunset-2h30
- 17:00

Examples for invalid timespecs:

- noon (undefined)
- 17m (plain offset)
- sunrise + 1m (spaces not allowed)

15.2.2. Interfaces and Classes

15.2.2.1. Timer

```
1 interface Timer
```

Represents a timer, as generated by any of the timer functions. A reference to it may be kept to later cancel the timer. If cancellation of the timer is not needed, it can simply be ignored.

```
1 function cancel()
```

Cancel the referenced timer, if it is still active.

```
1 function isActive()
```

[1.7.3]

Returns:

true If the timer is active

false If the timer triggered or was manually cancelled

15.3. Zigbee API

```
1 var Zigbee = require('sys/zigbee');
```

15.3.1. Exported functions

15.3.1.1. onReady()

```
1 function onReady(callback)
```

Register a callback function to be invoked when the Zigbee module is ready and devices can be looked up.

Parameters:

callback The function to invoke when the ZigBee module is ready. Will be invoked without any parameters.

Throws:

TypeError The callback parameter is not callable

The required Zigbee devices shall be looked-up via `getDevice()` from within the callback or after the callback was invoked. Device lookup will fail if attempted before the

15.3.1.2. onUpdate()

```
1 function onUpdate(callback)
```

Register a callback function to be invoked when an update of the internal representation of the Zigbee devices was received. Under normal circumstances and for simple cases, this event does not need to be handled. Any in-memory representation will be updated seamlessly, while e.g. removed devices are invalidated and would trigger an error on access.

Parameters:

callback The function to invoke on an update. Will be invoked without any parameters.

Throws:

TypeError The callback parameter is not callable

15.3.1.3. `getDevice ()`

```
1 function getDevice(address)
```

Parameters:

address Specifies the 64-bit IEEE address of the device as a string. The address must be given in hexadecimal, with or without double-colon separators (e.g. '001fee0012345678' or '00:1f:ee:00:12:34:56:78').

Throws:

SyntaxError The address was invalid and could not be parsed.

Returns:

The Device instance (see [Section 15.3.2.1](#)) or undefined if the device was not found.

15.3.1.4. `getGroupByName()`

[1.9.4]

```
1 function getGroupByName(name)
```

Parameters:

name Specifies the user-visible name of the group.

Returns:

The Group instance (see [Section 15.3.2.3](#)) or **undefined** if the group was not found. The group name is configured when the group is created via the ubisys app. If multiple groups exist with the same name, it is implementation-defined, which one is returned.

15.3.1.5. `getGroupById()`

[1.9.4]

```
1 function getGroupById(id)
```

Parameters:

id Specifies the internal group id. The group id remains constant if the group is renamed or modified, but not if it is deleted and re-created.

Throws:

TypeError id is not an integer number

RangeError id is negative

Returns:

The Group instance (see [Section 15.3.2.3](#)) or **undefined** if the group does not exist.

15.3.1.6. `getGroupByAddress()`

[1.9.4]

```
1 function getGroupByAddress(address)
```

Parameters:

address Specifies the group address.

Throws:

TypeError address is not an integer number

RangeError address is not in the range [1, 0xffff]

Returns:

The Group instance (see [Section 15.3.2.3](#)) or **undefined** if the group does not exist. If multiple groups exists with the same group address (in different networks), it is implementation-defined, which group will be returned.

15.3.1.7. `getScene()`

```
1 function getScene(name)
```

Parameters:

name Specifies the name of the scene.

Throws:

TypeError name is not a string

Returns:

The Scene instance (see [Section 15.3.2.4](#)) or undefined if the scene does not exist. The scene name is set when the scene is created via the ubisys app.

15.3.1.8. `getSceneById()`

```
1 function getSceneById(id)
```

Parameters:

id Specifies the internal id of the scene. The id is assigned on scene creation and remains stable if the scene is renamed or modified.

Throws:

TypeError id is not an integer

Returns:

The Scene instance (see [Section 15.3.2.4](#)) or undefined if the scene does not exist.

15.3.1.9. getScenes()

[1.7.3]

```
1 function getScenes()
```

Returns:

An array containing Scene (see [Section 15.3.2.4](#)), representing all defined scenes in the system.

15.3.1.10. lookup()

[1.9.4]

Provides a generic way to lookup Zigbee instances by a specification string.

```
1 function lookup(spec)
```

Parameters:

spec Specifies the instance to be retrieved.

Throws:

TypeError spec is not a string

SyntaxError spec has an invalid syntax which could not be parsed

RangeError a component of the spec string is outside of its valid range

Returns:

The specified instance or **undefined** if the requested instance does not exist.

Supported lookup specification:

- Device Lookup Spec string format: hexa-decimal address, e.g. 00:1f:ee:00:11:22:33:44
Returns a Device instance or **undefined** if the device does not exist.
- Combined device/application lookup

[2.0.3]

Spec string format: hexa-decimal address, followed by a hash sign (#), followed by the decimal endpoint number, e.g. 00:1f:ee:00:11:22:33:44#3. Returns an Application instance or **undefined** if the application does not exist

- Group lookup Spec string format: prefix "g:", followed by the group name, e.g. g:Entrance Lights. The group name is set when the group is created via the ubisys app. Returns a Group instance or **undefined** if the group does not exist.
- Group lookup by group ID Spec string format: prefix "g#:", followed by the group identifier, e.g. g#:17. The group ID is an internal identifier assigned on group creation. It remains stable across group renames or modifications (unless the group is deleted and recreated), Returns a Group instance or **undefined** if the group does not exist.
- Scene lookup Spec string format: prefix "s:", followed by the scene name, e.g. s:Entrance Lights Off. The scene name is set when the scene is created via the ubisys app. Returns a Scene instance or **undefined** if the scene does not exist.
- Scene lookup by scene ID Spec string format: prefix "s#:", followed by the scene identifier, e.g. s#:7. The scene ID is an internal identifier assigned on scene creation. It remains stable across scene renames or modifications (unless the scene is deleted and recreated), Returns a Scene instance or **undefined** if the scene does not exist.

15.3.2. Interfaces and classes

15.3.2.1. Device

Represents a Zigbee device.

```
1 function getApplication(endpoint)
```

Throws:

TypeError endpoint is not an integer

RangeError endpoint is out of range (1..240)

Returns:

The Application instance on the specific endpoint.

```
1 function getIEEEAddress()
```

Returns:

The 64-bit IEEE address as a string, with bytes separated by colons, e.g '00:1f:ee:00:11:22:33:44'.

```
1 function getShortAddress()
```

Returns:

The 16-bit short address as an integer.

```
1 function getManufacturer()
```

Returns:

The manufacturer string, as specified in the Basic cluster.

```
1 function getVendor()
```

Returns:

The vendor string, as specified in the Basic cluster.

```
1 function getModel()
```

Returns:

The model string, as specified in the Basic cluster.

```
1 function type()
```

[1.9.4]

Returns: The fixed string "device".

15.3.2.2. Application

```
1 function getProfile()
```

Returns:

The numerical profile identifier for this application.

```
1 function getType()
```

Returns:

The numerical type identifier for this application.

```
1 function getCluster(clusterId)
```

Obtain the cluster instance with the numerical clusterId.

Parameters:

clusterId The numeric cluster id

Throws:

TypeError If the clusterId is not an integer

RangeError If the clusterId is not within the valid range (0..0xffff)

Returns:

The Cluster instance or **undefined** if the cluster is not known. Note that the basic cluster (0) is currently not available.

```
1 function getOnOffCluster()
```

A convenience alias for getCluster(6).

```
1 function getLevelControlCluster()
```

A convenience alias for getCluster(8).

```
1 function getWindowCoveringCluster()
```

A convenience alias for getCluster(0x0102).

```
1 function getClientCluster(clusterId)
```

Obtain the client cluster instance with the numerical clusterId.

Parameters:

clusterId The numeric cluster id

Throws:

TypeError If the clusterId is not an integer

RangeError If the clusterId is not within the valid range (0..0xffff)

Returns:

The ClientCluster instance or **undefined** if the cluster is not known.

```
1 function getOnOffClientCluster()
```

Convenience alias for getClientCluster(0x0006).

```
1 function getLevelControlClientCluster()
```

Convenience alias for `getClientCluster(0x0008)`.

```
1 function getUbisysManagedInputClientCluster()
```

Convenience alias for `getClientCluster(0xfc02)`.

```
1 function type()
```

[1.9.4]

Returns:

The fixed string “application”.

15.3.2.3. Group

[1.9.4]

Represents a Zigbee group.

```
1 function toString()
```

Returns:

A string representation of this group. Mainly used for debugging/logging. The exact format is implementation-defined.

```
1 function getDescription()
```

Returns:

The user-assigned name/description of this group.

```
1 function id()
```

Returns:

The internal group id. Assignment is implementation-defined. The id remains stable if the group is renamed, but not if the group is removed and re-created.

```
1 function address()
```

Returns:

The group address of this group.

```
1 function type()
```

Returns:

The fixed string "group".

```
1 function members()
```

Returns:

An array of Application instances which represent the group members.

```
1 function getCluster(clusterId)
```

Obtain the cluster instance with the numerical clusterId.

Parameters:

clusterId The numeric cluster id

Throws:

TypeError If the clusterId is not an integer

RangeError If the clusterId is not within the valid range (0..0xffff)

Returns:

The cluster instance or **undefined** if the cluster is not known. Note that due to the nature of groups, the returned cluster instance only supports issuing commands, not retrieving or monitoring attributes, as they are tied to a specific device/application instance.

```
1 function getOnOffCluster()
```

A convenience alias for getCluster(6).

```
1 function getLevelControlCluster()
```

A convenience alias for getCluster(8).

```
1 function getWindowCoveringCluster()
```

A convenience alias for getCluster(0x0102).

15.3.2.4. Scene

Represents a Zigbee scene.

```
1 function recall(transitionTime)
```

Recalls the scene, i.e. requests to set all scene attributes to there stored state.

Parameters:

transitionTime Optional. The transition time to use when applying the scene in seconds (internal resolution is 100ms).

```
1 function toString()
```

Returns:

A string representation of this scene. Mainly used for debugging/logging. The exact format is implementation-defined.

```
1 function type()
```

[1.9.4]

Returns:

The fixed string "scene".

```
1 function getDescription()
```

Returns:

The user-assigned name/description of this scene.

15.3.2.5. Cluster

```
1 function command(command, payload, callback)
```

Send an arbitrary ZCL command to the cluster. Can be used to assemble custom commands for which no wrappers are available. Only commands directed to the cluster are supported. No profile-wide commands and no manufacturer-specific commands are supported.

Parameters:

command The numerical command to send. The meaning of the command is defined by the cluster.

payload The command payload to send. Optional. May be any of the supported buffer instances (see [Section 15.4](#)).

callback The callback to invoke on completion of the command. Will be invoked with a ZigBeeStatus instance (see 0) as the only parameter.

Throws:

TypeError Invalid types for command, payload or callback.

RangeError command out of range

```
1 function getAttribute(attributeId)
```

Obtains a cached attribute. The set of cached attributes is specified in [Section 15.3.3](#))

Parameters:

attributeld The numerical Id of the attribute

Throws:

TypeError attributeld is not an integer

RangeError attributeld is out of range (0..0xffff)

Returns:

The attribute instance or undefined if the attribute was not found.

```
1 function onAttributeChanged(attributeId, callback)
```

Registers a function to be called whenever the given attribute changes. Note that this only works for reportable attributes and for devices which support reporting (see [Section 15.3.3](#)). The function will be invoked with the `attributeld` as the first parameter, an attribute instance (5.2.2.7) as the second parameter.

Parameters:

attributeld The numerical Id of the attribute

callback The function to invoke

Throws:

TypeError attributeld is not an integer or callback is not a function

RangeError attributeld is out of range (0..0xffff)

```
1 function readAttribute(attributeId, callback)
```

Sends a ZCL “Read Attributes” request to the cluster and delivers the result to the callback function. The callback function will be invoked with a `ZigBeeStatus` instance as the first parameter and, on success, with an attribute instance as the second parameter.

Parameters:

- attributeld** The numerical Id of the attribute
- callback** The function to invoke to deliver the result

Throws:

- TypeError** attributeld is not an integer or callback is not a function
- RangeError** attributeld is out of range (0..0xffff)

```
1 function writeAttribute(attributeId, type, data, callback)
```

Sends a ZCL "Write Attribute" request to the cluster and delivers the result to the callback function. The callback function will be invoked with a ZigBeeStatus instance as the only parameter.

Parameters:

- attributeld** The numerical Id of the attribute
- type** The type of the attribute to write; either numeric or as a string alias
- data** The data to write
- callback** The function to invoke to deliver the result

Throws:

- TypeError** If the attributeld is not an integer or the callback is not callable; if the type is neither a number nor a string
- RangeError** If the attributeld is not within the valid range (0..0xffff) or if the numerical attribute type is not within the valid range (0..0xff)
- Error** If the attribute data could not be parsed or the attribute type alias is unknown.

Supported types and payloads are as follows:

Type Alias	ZCL type	Payload
data8	0x08	hex string of matching length. Alternatively any Buffer object.
data16	0x09	
data24	0x0a	
data32	0x0b	
data40	0x0c	
data48	0x0d	
data56	0x0e	
data64	0x0f	
bool, boolean	0x10	
bitmap8	0x18	Either a number, a hex string or a binary string. For the string representation, the number of digits must match the bitmap size, e.g. 16 binary digits or four hex digits for bitmap16.
bitmap16	0x19	
bitmap24	0x1a	
bitmap32	0x1b	
bitmap40	0x1c	
bitmap48	0x1d	
bitmap56	0x1e	
bitmap64	0x1f	
unsigned8	0x20	Number. Be aware of the limited precision JavaScript offers for unsigned56 and unsigned64 (internal double-precision floating point representation). Consider using a Buffer object instead.
unsigned16	0x21	
unsigned24	0x22	
unsigned32	0x23	
unsigned40	0x24	
unsigned48	0x25	
unsigned56	0x26	
unsigned64	0x27	
signed8	0x28	
signed16	0x29	
signed24	0x2a	
signed32	0x2b	
signed40	0x2c	
signed48	0x2d	
signed56	0x2e	
signed64	0x2f	
enum8, enumeration8	0x30	Number
enum16, enumeration16	0x31	
raw, rawdata	0x41	hex string or any Buffer object
array	0x48	

The type may be specified as a number as well. The payload for any type may be given as a Buffer object. Valid buffer types are Duktape buffers, Node.js-like buffers and ES2015 ArrayBuffers, e.g. an Uint8Array. Please refer to section [Section 15.4](#).

Be aware that when passing a Buffer object, no constraints will be enforced. This may result in invalid ZCL requests being sent, e.g. a write request for an Unsigned8 with a payload of multiple bytes.

By specifying the type as a numeric value and the payload as a Buffer, any ZCL attribute type can be written; not only those listed in the table above.

```
1 function type()
```

Returns:

The fixed string “cluster”.

15.3.2.6. OnOffCluster

Specialization of cluster for the On/Off cluster. Provides all functions of the generic cluster, augmented by specialized functions for the On/Off cluster.

```
1 function off(callback)
```

Sends an “off” command to the cluster.

Parameters:

callback Optional. The function to invoke on completion of this command. Will be invoked with a ZigBeeStatus instance as its parameter.

Throws:

TypeError callback is specified, but not a function

```
1 function on(callback)
```

Sends an “on” command to the cluster.

Parameters:

callback Optional. The function to invoke on completion of this command. Will be invoked with a ZigBeeStatus instance as its parameter.

Throws:

TypeError callback is specified, but not a function

```
1 function toggle(callback)
```

Sends an “toggle” command to the cluster.

Parameters:

callback Optional. The function to invoke on completion of this command. Will be invoked with a ZigBeeStatus instance as its parameter.

Throws:

TypeError callback is specified, but not a function

```
1 function onOnOffChanged(callback)
```

Registers a function to be called when the On/Off state changes. Called with a bool value as its only parameter or undefined if the status is not available.

Throws:

TypeError callback is not a function

```
1 function isOn()
```

Returns:

The current On/Off status as a bool (true/false) or undefined if not available.

15.3.2.7. LevelControlCluster

Specialization of cluster for the Level Control cluster.

```
1 function moveToLevel(level, ttime, callback)
```

Moves to the specified level with the specified transition time.

Parameters:

level The level to move to (0..254)

ttime Transition time in units of 0.1 seconds.

callback Optional. The function to invoke on completion of this command. Will be invoked with a ZigBeeStatus instance as its parameter.

Throws:

TypeError level/ttime are not integers. callback is specified, but not callable.

RangeError level (0..254) or ttime (0..65535)

```
1 function stepDown(stepSize, ttime, callback)
```

Steps down by the specified step size in the specified transition time.

Parameters:

- stepSize** The step size (0..254)
- ttime** Transition time in units of 0.1 seconds.
- callback** Optional. The function to invoke on completion of this command. Will be invoked with a ZigBeeStatus instance as its parameter.

Throws:

- TypeError** level/ttime are not integers. callback is specified, but not callable.
- RangeError** stepSize (0..254) or ttime (0..65535)

```
1 function stepUp(stepSize, ttime, callback)
```

Steps up by the specified step size in the specified transition time.

Parameters:

- stepSize** The step size (0..254)
- ttime** Transition time in units of 0.1 seconds.
- callback** Optional. The function to invoke on completion of this command. Will be invoked with a ZigBeeStatus instance as its parameter.

Throws:

- TypeError** level/ttime are not integers. callback is specified, but not callable.
- RangeError** stepSize (0..254) or ttime (0..65535)

```
1 function stop(callback)
```

Stops any pending move to level, move or step command.

Parameters:

- callback** Optional. The function to invoke on completion of this command. Will be invoked with a ZigBeeStatus instance as its parameter.

Throws:

- TypeError** callback is specified, but not callable.

```
1 function moveToLevelWithOnOff(level, ttime, callback)
```

Same as `moveToLevel()`, but affects the On/Off state of any linked On/Off cluster.

```
1 function stepDownWithOnOff(stepSize, ttime, callback)
```

Same as `stepDown()`, but affects the On/Off state of any linked On/Off cluster.

```
1 function stepUpWithOnOff(stepSize, ttime, callback)
```

Same as `stepUp()`, but affects the On/Off state of any linked On/Off Cluster.

```
1 function onLevelChanged(callback)
```

Registers a function to be called when the level changes. The function will be invoked with a numeric level as the parameter, or undefined if unknown or invalid.

Parameters:

callback The function to invoke.

Throws:

TypeError callback is not callable.

```
1 function getLevel()
```

Returns:

The current level as an integer or undefined if not available.

15.3.2.8. WindowCoveringCluster

Specialization of cluster for the Window Covering cluster.

```
1 function up(callback)
```

Starts moving the Window Covering device towards the up/open position.

Parameters:

callback Optional. The function to invoke to deliver the result of the operation.

Throws:

TypeError callback is specified, but not callable.

```
1 function open(callback)
```

Alias for up().

```
1 function down(callback)
```

Starts moving the Window Covering device towards the down/closed position.

Parameters:

callback Optional. The function to invoke to deliver the result of the operation.

Throws:

TypeError callback is specified, but not callable.

```
1 function close(callback)
```

Alias for down().

```
1 function stop(callback)
```

Stops the movement of the Window Covering device.

Parameters:

callback Optional. The function to invoke to deliver the result of the operation.

Throws:

TypeError callback is specified, but not callable.

```
1 function gotoLiftValue(liftValue, callback)
```

Moves the Window Covering device to the specified lift value (in centimeters). The lift value must be in the range given by the `InstalledOpenLimitLift` (0x0010) and `InstalledClosedLimitLift` (0x0011) attributes. If the value is outside this range, the device shall report an error, which will be delivered to the callback function as a `ZigBeeStatus` instance representing the `ZCL:INVALID_VALUE` error. No `RangeError` is thrown in this case, as the verification is done on the device. This is an optional command, which may not be implemented by all devices. The device shall report an unsupported command and the callback would be invoked with a `ZigBeeStatus` instance representing the failure (`ZCL: UNSUP_CLUSTER_COMMAND`).

Parameters:

- liftValue** The lift value to move to.
- callback** Optional. The function to invoke to deliver the result of the operation.

Throws:

- TypeError** callback is specified, but not callable; liftValue is not an integer.
- RangeError** liftValue is outside the valid range (0..0xffff)

```
1 function gotoLiftPercentage(liftPercentage, callback)
```

Moves the Window Covering device to the specified lift percentage. This is an optional command, which may not be implemented by all devices. The device shall report an unsupported command and the callback would be invoked with a ZigBeeStatus instance representing the failure (ZCL: UNSUP_CLUSTER_COMMAND).

Parameters:

- liftPercentage** The lift percentage to move to.
- callback** Optional. The function to invoke to deliver the result of the operation.

Throws:

- TypeError** callback is specified, but not callable; liftPercentage is not an integer.
- RangeError** liftPercentage is outside the valid range (0..100)

```
1 function gotoTiltValue(tiltValue, callback)
```

Moves the Window Covering device to the specified tilt value. The lift value must be in the range given by the InstalledOpenLimitTilt (0x0012) and InstalledClosedLimitTilt (0x0013) attributes. If the value is outside this range, the device shall report an error, which will be delivered to the callback function as a ZigBeeStatus instance representing the ZCL:INVALID_VALUE error. No RangeError is thrown in this case, as the verification is done on the device. This is an optional command, which may not be implemented by all devices. The device shall report an unsupported command and the callback would be invoked with a ZigBeeStatus instance representing the failure (ZCL: UNSUP_CLUSTER_COMMAND).

Parameters:

- tiltValue** The lift value to move to (0..900), in units of 0.1 degrees, corresponding to 0..90 degrees.
- callback** Optional. The function to invoke to deliver the result of the operation.

Throws:

- TypeError** callback is specified, but not callable; liftValue is not an integer.
- RangeError** liftValue is outside the valid range (0..0xffff)

```
1 function gotoTiltPercentage(tiltPercentage, callback)
```

Moves the Window Covering device to the specified tilt percentage. This is an optional command, which may not be implemented by all devices. The device shall report an unsupported command and the callback would be invoked with a ZigBeeStatus instance representing the failure (ZCL: UNSUP_CLUSTER_COMMAND).

Parameters:

- tiltPercentage** The lift percentage to move to.
- callback** Optional. The function to invoke to deliver the result of the operation.

Throws:

- TypeError** callback is specified, but not callable; tiltPercentage is not an integer.
- RangeError** tiltPercentage is outside the valid range (0..100)

15.3.2.9. ClientCluster

Represents a zigbee client cluster and allows registering callback functions for inbound commands.

```
1 function type()
```

Returns the fixed string "clientcluster".

```
1 function on(command, callback)
```

Registers a handler for the given command. The command may be given as a numeric value. For several clusters, string aliases are defined as per the table below. Handlers for manufacturer-specific commands must be given as a string in the form "manufacturer:command", whereas both manufacturer and command are to be encoded as hexadecimal, e.g. "10f2:1". The ubisys manufacturer code **10f2** may be given as "ubisys" as well.

The callback will be invoked with the command payload, which is provided as an object. The object always contains the field **raw**, which contains the unparsed command payload as bytes, stored in a Buffer object. For certain commands (as per table below), the payload is parsed and provided as a parsed representation.

Throws:

- TypeError** callback is not callable

- RangeError** command is out of the allowed range (0..255)

- Error** unknown command alias

The following command aliases are defined and payloads are parsed:

Cluster	Command	Parsed payload fields
On/Off	off	-
	on	-
	toggle	-
Level Control	move-to-level	level
		transition-time
		options-mask
		options-override
	move	move-mode
		rate
		options-mask
		options-override
	step	step-mode
		step-size
		transition-time
		options-mask
	stop	options-override
		level
		transition-time
		options-mask
	move-to-level-with-on-off	options-override
		move-mode
		rate
		options-mask
move-with-on-off	options-override	
	step-mode	
	step-size	
	transition-time	
step-with-onoff	options-mask	
	options-override	
	stop-with-onoff	
	position	
ubisys managed input	ubisys:switch-latched	position
	ubisys:initial-press	position
	ubisys:long-press	position
	ubisys:short-release	position

Cluster	Command	Parsed payload fields
	ubisys:long-release	position

15.3.2.10. Attribute

Supported data types (with mappings to Java types):

- Null
- Bool
- Unsigned integer (all widths)
- Signed integer (all widths)
- CharacterString
- Bitmap (all widths)

More types will be added in a future revision.

```
1 function id()
```

Returns:

The numerical Id of the represented attribute.

```
1 function type()
```

Returns:

The decoded, human-readable attribute type as a string.

```
1 function rawType()
```

Returns:

The numerical type of the attribute.

```
1 function value()
```

Returns:

The decoded attribute value or undefined if this type is not (yet) supported.

Null is mapped to null. Bool is mapped to true/false or undefined (for the invalid value). Signed/unsigned integer types are mapped to numbers or undefined (for the invalid value). A CharacterString is mapped to a String. A Bitmap will be mapped to a String consisting of binary digits for each bit. For bitmap types, the testBit() function is provided.

```
1 function toString()
```

Returns:

String representation of this attribute. Mainly for debugging/logging.


```
1 function testBit(index)
```

Parameters:

index The bit index to test

Returns:

true if the bit is set, false if unset.

Throws:

TypeError index is not an integer or the attribute is not of a Bitmap type

RangeError index exceeds the Bitmap width

15.3.2.11. ZigbeeStatus

Represents the outcome of a ZigBee/ZCL Request. Passed to the callback functions for a ZCL Request.

```
1 function success()
```

Returns:

true if the operation was successful, false otherwise.

```
1 function toString()
```

Returns:

A string representation with a brief error description or "success".

15.3.3. Cached Attributes

The following attributes are available via `getAttribute()` in the cluster interface. Attributes marked as “reportable” are configured for reporting. The gateway will therefore automatically receive updates for these attribute. Notifications can be requested via the `onAttributeChanged` function in cluster. Please not that there are devices (e.g. Philips Hue) which do not support reporting. Attributes marked as “static” are usually read once and not updated automatically.

15.3.3.1. Power Configuration Cluster (0x0001)

Id	Name	Type
0x0000	Mains Voltage	static
0x0001	Mains Frequency	static
0x0010	Mains Alarm Mask	static
0x0020	Battery Voltage	static
0x0021	Battery Percentage Remaining	reported
0x0031	Battery Size	static

Id	Name	Type
0x0033	Battery Quantity	static
0x0034	Battery Rated Voltage	static
0x0035	Battery Alarm Mask	static
0x0036	Battery Voltage Minimum Threshold	static
0x0037	Battery Voltage Threshold 1	static
0x0038	Battery Voltage Threshold 2	static
0x0039	Battery Voltage Threshold 3	static
0x003a	Battery Percentage Minimum Threshold	static
0x003b	Battery Percentage Threshold 1	static
0x003c	Battery Percentage Threshold 2	static
0x003d	Battery Percentage Threshold 3	static
0x003e	Battery Alarm State	reported

15.3.3.2. On/Off Cluster (0x0006)

Id	Name	Type
0x0000	On/Off	reported
0x4003	Start-Up On/Off	static

15.3.3.3. On/Off Switch Configuration (0x0007)

Id	Name	Type
0x0000	Switch Type	static
0x0000	Switch Type	static
0x0010	Switch Actions	static
0x0020	Multiple Switch Function Profile	static
0x0021	Multiple Switch Profile Parameter 1	static
0x0022	Multiple Switch Profile Parameter 2	static
0x0023	Multiple Switch Profile Parameter 3	static

15.3.3.4. Level Control Cluster (0x0008)

Id	Name	Type
0x0000	Current Level	reported
0x000f	Options	static
0x0010	On/Off Transition Time	static
0x0011	On Level	static
0x0012	On Transition Time	static
0x0013	Off Transition Time	static
0x0014	Default Move Rate	static
0x4000	Start-Up Current Level	static

15.3.3.5. OTA Upgrade (0x0019)

Id	Name	Type
0x0000	Upgrade Server ID	static
0x0001	File Offset	static
0x0002	Current File Version	static
0x0003	Current ZigBee Stack Version	static
0x0007	Manufacturer ID	static
0x0008	Image Type ID	static

15.3.3.6. Poll Control (0x0020)

Id	Name	Type
0x0000	Check-In Interval	static
0x0001	Long Poll Interval	static
0x0002	Short Poll Interval	static
0x0003	Fast Poll Timeout	static
0x0004	Check-In Interval Minimum	static
0x0005	Long Poll Interval Minimum	static
0x0006	Fast Poll Timeout Maximum	static

15.3.3.7. Door Lock (0x0101)

Id	Name	Type
0x0000	Lock State	reported
0x0001	Lock Type	static
0x0002	Actuator Enabled	static
0x0003	Door State	reported
0x0004	Door Open Events	static
0x0005	Door Closed Events	static
0x0006	Open Period	static
0x0010	Number Of Log Records Supported	static
0x0011	Number Of Total Users Supported	static
0x0012	Number Of PIN Users Supported	static
0x0013	Number Of RFID Users Supported	static
0x0014	Number Of Week Day Schedules Supported Per User	static
0x0015	Number Of Year Day Schedules Supported Per User	static
0x0016	Number Of Holiday Schedules Supported	static
0x0017	Maximum PIN Code Length	static
0x0018	Minimum PIN Code Length	static
0x0019	Maximum RFID Code Length	static
0x001a	Minimum RFID Code Length	static
0x0020	Enable Logging	static
0x0021	Language	static
0x0022	LED Settings	static
0x0023	Auto Relock Time	static
0x0024	Sound Volume	static
0x0025	Operating Mode	reported
0x0026	Supported Operating Mode	static
0x0027	Default Configuration Register	static
0x0028	Enable Local Programming	static
0x0029	Enable One Touch Locking	static
0x002a	Enable Inside Status LED	static
0x002b	Enable Privacy Mode Button	static
0x0030	Wrong Code Entry Limit	static
0x0031	User Code Temporary Disable Time	static
0x0032	Send PIN Over The Air	static
0x0033	Require PIN For RF Operation	static
0x0034	Security Level	reported
0x0040	Alarm Mask	static
0x0041	Keypad Operation Event Mask	static
0x0042	RF Operation Event Mask	static
0x0043	Manual Operation Event Mask	static
0x0044	RFID Operation Event Mask	static
0x0045	Keypad Programming Event Mask	static
0x0046	RF Programming Event Mask	static
0x0047	RFID Programming Event Mask	static

15.3.3.8. Window Covering Cluster (0x0102)

Id	Name	Type
0x0000	Window Covering Type	static
0x0000	Window Covering Type	static
0x0007	Config/Status	static

Id	Name	Type
0x0007	Config/Status	static
0x0008	Current Position Lift Percentage	reported
0x0009	Current Position Tilt Percentage	reported
0x000a	Operational Status	reported
0x0010	Installed Open Limit - Lift	static
0x0010	Installed Open Limit - Lift	static
0x0011	Installed Closed Limit - Lift	static
0x0011	Installed Closed Limit - Lift	static
0x0012	Installed Open Limit - Tilt	static
0x0012	Installed Open Limit - Tilt	static
0x0013	Installed Closed Limit - Tilt	static
0x0013	Installed Closed Limit - Tilt	static
0x0017	Mode	static
0x1000	Turn-Around Guard Time	static
0x1001	Lift-To-Tilt Transition Step 1	static
0x1002	Total Step 1	static
0x1003	Lift-To-Tilt Transition Step 2	static
0x1004	Total Step 2	static
0x1005	Additional Steps	static

15.3.3.9. Thermostat (0x0201)

Id	Name	Type
0x0000	Local Temperature	reported
0x0001	Outdoor Temperature	static
0x0002	Occupancy	reported
0x0003	Absolute Minimum Heating Setpoint Limit	static
0x0004	Absolute Maximum Heating Setpoint Limit	static
0x0005	Absolute Minimum Cooling Setpoint Limit	static
0x0006	Absolute Maximum Cooling Setpoint Limit	static
0x0007	PI Cooling Demand	reported
0x0008	PI Heating Demand	reported
0x0009	HVAC System Type Configuration	static
0x0010	Local Temperature Calibration	static
0x0011	Occupied Cooling Setpoint	reported
0x0012	Occupied Heating Setpoint	reported
0x0013	Unoccupied Cooling Setpoint	reported
0x0014	Unoccupied Heating Setpoint	reported
0x0015	Minimum Heating Setpoint Limit	static
0x0016	Maximum Heating Setpoint Limit	static
0x0017	Minimum Cooling Setpoint Limit	static
0x0018	Maximum Cooling Setpoint Limit	static
0x0019	Minimum Setpoint Dead Band	static
0x001a	Remote Sensing	static
0x001b	Control Sequence Of Operation	static
0x001c	System Mode	reported
0x001d	Alarm Mask	static
0x001e	Thermostat Running Mode	reported
0x0034	Occupied Setback	static
0x0035	Occupied Setback Minimum	static
0x0036	Occupied Setback Maximum	static

Id	Name	Type
0x0037	Unoccupied Setback	static
0x0038	Unoccupied Setback Minimum	static
0x0039	Unoccupied Setback Maximum	static
0x003a	Emergency Heat Delta	static

15.3.3.10. Fan Control (0x0202)

Id	Name	Type
0x0000	Fan Mode	reported
0x0001	Fan Mode Sequence	static

15.3.3.11. Color Control Cluster (0x0300)

Id	Name	Type
0x0000	Current Hue	reported
0x0001	Current Saturation	reported
0x0003	Current X	reported
0x0004	Current Y	reported
0x0007	Current Temperature Mireds	reported
0x0008	Color Mode	static
0x000f	Options	static
0x0010	Number Of Primaries	static
0x0011	Primary 1X	static
0x0012	Primary 1Y	static
0x0013	Primary 1 Intensity	static
0x0015	Primary 2X	static
0x0016	Primary 2Y	static
0x0017	Primary 2 Intensity	static
0x0019	Primary 3X	static
0x001a	Primary 3Y	static
0x001b	Primary 3 Intensity	static
0x0020	Primary 4X	static
0x0021	Primary 4Y	static
0x0022	Primary 4 Intensity	static
0x0024	Primary 5X	static
0x0025	Primary 5Y	static
0x0026	Primary 5 Intensity	static
0x0028	Primary 6X	static
0x0029	Primary 6Y	static
0x002a	Primary 6 Intensity	static
0x4000	Enhanced Current Hue	static
0x4001	Enhanced Color Mode	static
0x4002	Color Loop Active	static
0x4003	Color Loop Direction	static
0x4004	Color Loop Time	static
0x4005	Color Loop Start Enhanced Hue	static
0x4006	Color Loop Stored Enhanced Hue	static
0x400a	Color Capabilities	static
0x400b	Color Temperature Physical Minimum Mireds	static
0x400c	Color Temperature Physical Maximum Mireds	static
0x400d	Couple Color Temperature To Level Minimum Mireds	static

Id	Name	Type
0x4010	Start-Up Color Temperature Mireds	static

15.3.3.12. Ballast Configuration (0x0301)

Id	Name	Type
0x0000	Physical Minimum Level	static
0x0001	Physical Maximum Level	static
0x0010	Minimum Level	static
0x0011	Maximum Level	static

15.3.3.13. Illuminance Measurement (0x0400)

Id	Name	Type
0x0000	Measured Value	reported
0x0001	Minimum Measured Value	static
0x0002	Maximum Measured Value	static
0x0003	Tolerance	static
0x0004	Light Sensor Type	static

15.3.3.14. Illuminance Level Sensing (0x0401)

Id	Name	Type
0x0000	Level Status	reportable
0x0001	Light Sensor Type	static
0x0010	Illuminance Target Level	static

15.3.3.15. Temperature Measurement (0x402)

Id	Name	Type
0x0000	Measured Value	reported
0x0001	Minimum Measured Value	static
0x0002	Maximum Measured Value	static
0x0003	Tolerance	static

15.3.3.16. Relative Humidity Measurement (x0405)

Id	Name	Type
0x0000	Measured Value	reported
0x0001	Minimum Measured Value	static
0x0002	Maximum Measured Value	static
0x0003	Tolerance	static

15.3.3.17. Occupancy Sensing (0x0406)

Id	Name	Type
0x0000	Occupancy	reported
0x0001	Occupancy Sensor Type	static
0x0002	Occupancy Sensor Type Bitmap	static
0x0010	PIR Occupied To Unoccupied Delay	static
0x0011	PIR Unoccupied To Occupied Delay	static
0x0012	PIR Unoccupied To Occupied Threshold	static
0x0020	Ultrasonic Occupied To Unoccupied Delay	static

Id	Name	Type
0x0021	Ultrasonic Unoccupied To Occupied Delay	static
0x0022	Ultrasonic Unoccupied To Occupied Threshold	static
0x0030	Physical Contact Occupied To Unoccupied Delay	static
0x0031	Physical Contact Unoccupied To Occupied Delay	static
0x0032	Physical Contact Unoccupied To Occupied Threshold	static

15.3.3.18. Leaf Wetness (0x0407)

Id	Name	Type
0x0000	Measured Value	reported
0x0001	Minimum Measured Value	static
0x0002	Maximum Measured Value	static
0x0003	Tolerance	static

15.3.3.19. Soil Moisture (0x0408)

Id	Name	Type
0x0000	Measured Value	reported
0x0001	Minimum Measured Value	static
0x0002	Maximum Measured Value	static
0x0003	Tolerance	static

15.3.3.20. IAS Zone Cluster (0x0500)

Id	Name	Type
0x0000	Zone State	static
0x0001	Zone Type	static
0x0002	Zone Status	reported
0x0010	IAS CIE Address	static
0x0011	Zone ID	static
0x0012	Number Of Zone Sensitivity Levels Supported	static
0x0013	Current Zone Sensitivity Level	static

15.3.3.21. IAS WD Cluster (0x0502)

Id	Name	Type
0x0000	Max Duration	static

15.3.3.22. Metering (0x0702)

Id	Name	Type
0x0000	Current Summation Delivered	reported
0x0200	Meter Status	static
0x0300	Unit Of Measure	static
0x0301	Multiplier	static
0x0302	Divisor	static
0x0306	Metering Device Type	static
0x0400	Instantaneous Demand	reported

15.3.3.23. Device Setup (0xfc00)

Id	Name	Type
0x0000	Input Configurations	static
0x0001	Input Actions	static

15.3.3.24. Dimmer Setup (0xfc01)

Id	Name	Type
0x0000	Capabilities	static
0x0001	Status	static
0x0002	Mode	static

15.4. Buffer Support

Duktape internally provides support for various buffer types. Any of those may be used where a buffer instance is required, e.g. `cluster.sendCommand()`. The most useful is probably the implementation resembling the `node.js Buffer` class.

The following example manually assembles the payload for a step command (2) to the Level Control cluster and sends the command to the cluster. The payload format is as following:

Size	1	1	2
Content	Step mode	Step size	Transition time
Value	0 (up)	10	10

```
1 // Allocate a buffer of 4 bytes
2 var buffer = new Buffer(4);
3
4 // Offset 0: Step mode: 0
5 buffer.writeUInt8(0, 0);
6 // Offset 1: Step size: 10
7 buffer.writeUInt8(10, 1);
8 // Offset 2: Transition time: 10 (16 bit)
9 buffer.writeUInt16LE(10, 2);
10
11 // Send the command and print the status
12 clusterLevel.command(2, buffer, function(status)
13 {
14     print('Result:', status);
15 });
```

References :

Node.js Buffer Object <https://nodejs.org/api/buffer.html>

Duktape Buffer APIs <https://github.com/svaarala/duktape/blob/master/doc/buffers.rst>

ES2015 ArrayBuffer https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer

15.5. Push Notification API

```
1 var Push = require('builtin/push');
```

Allows sending push notifications to mobile devices:

```
1 function submitNotification(message)
```

Parameters:

message The message to push to mobile devices

```
1 function submitNotification(message, title)
```

Parameters:

message The message to push to mobile devices

```
1 function onStatusChange(attributeID, attribute)
2 {
3     print('IAS Zone status changed to ', attribute);
4
5     if (attribute.testBit(0))
6     {
7         Push.submitNotification('Door contact reports', 'Main door opened');
8     }
9     else
10    {
11        Push.submitNotification('Door contact reports', 'Main door closed');
12    }
13 }
```

15.6. Global variables

```
1 var Globals = require('sys/globals');
```

Allows to store and retrieve gateway-global variables and to send messages to other scripts. This can e.g. be used to implement some kind of state synchronization and simple communication facilities between independent scripts on the same gateway.

No concept of ownership or access control exists, i.e. any script can write to any variable.

```
1 function set(name, value)
```

Parameters:

name The name of the global variable to set

value The value to set. This can be an arbitrary JavaScript value, e.g. a number, string, array or object. No functions can be stored.

```
1 function get(name)
```

Parameters:

name The name of the global variable to retrieve

Returns the value of the function or undefined if no such variable was set.

```
1 function onChange(name, callback)
```

Parameters:

name The name of the global variable to monitor

callback The function to invoke whenever the global variable changes. This will be invoked with the name and value as parameters.

Throws:

TypeError the callback is not callable

```
1 function send(topic, payload)
```

Allows sending of messages to specific topics. Any script listening to this topic will receive the message. A message payload can be of an arbitrary JavaScript type with the exception of functions, e.g. a string, number, bool or an object.

Parameters:

topic The topic to send the message to

payload The message payload

```
1 function listen(topic, callback)
```

Allows listening for messages on a specific topic. The callback function will be invoked for each message sent for this specific topic.

Parameters:

- topic** The topic to listen to
- payload** The callback to invoke. The callback will receive the message payload as a parameter.

Throws:

- TypeError** the callback is not callable

15.7. HTTP Client API

```
1 var Http = require('sys/http');
```

15.7.1. Exported functions

Allows a script to send HTTP requests to an external HTTP server.

```
1 function request()
```

Creates a new HttpRequest instance.

```
1 new FormData()
```

Creates a new FormData instance to create **multipart/form-data** POST payloads.

15.7.2. Classes

15.7.2.1. HttpRequest

Represents an outgoing Http request to be sent.

```
1 function method(string)
```

Allows to set the HTTP request method. Supported methods are **GET**, **POST**, **PUT** and **HEAD**. Defaults to **GET** if not invoked.

Throws:

- Error** An invalid request method was given.

- Error** The request was already sent.

```
1 function url(string)
```

Sets the request URL. Supported URI schemes are **http** and **https**.

Throws:

Error The request was already sent.

```
1 function userAgent(string)
```

Sets the user agent header to the given string.

Throws:

Error The request was already sent.

```
1 function failOnHttpError(bool)
```

If set to true, a HTTP 4xx or 5xx response will cause an error. Defaults to false, i.e. an HTTP response is always considered success, no matter the response status. In this case, the script has to explicitly check the HTTP response status.

Throws:

Error The request was already sent.

```
1 function header(header, value)
```

Sets an HTTP request header.

Throws:

Error The request was already sent.

```
1 function username(string)
```

Sets the username to be used for HTTP authentication. Supported authentication schemes are Basic, Digest and NTLM.

Throws:

Error The request was already sent.

```
1 function password()
```

Sets the password to be used for HTTP authentication. Supported authentication schemes are Basic, Digest and NTLM.

Throws:

Error The request was already sent.

```
1 function postData(data, type)
```

Sets the data to be sent as the POST request payload.

Valid values for type are:

- **raw**

The provided data is sent as-is. The data must either be of type string or Buffer.

- **formdata**

The data must be a key-value map and will be encoded as application/x-www-form-urlencoded. Alternatively, an already encoded string may be passed.

- **json**

The provided data is a JavaScript object to be encoded as JSON.

Throws:

Error The request was already sent.

Error postData() was invoked for a non-POST request.

TypeError a parameter is of an incorrect type.

```
1 function formData(data)
```

Provides a FormData instance to be sent as the POST request payload.

Throws:

Error The request was already sent.

Error formData() was invoked for a non-POST request.

TypeError The passed data is not a FormData instance

```
1 function verifyPeer(bool)
```

Set to false to disable verification of the server's TLS/x509 certificate. Defaults to true (i.e. verify). Only applicable to **https** URLs.

Throws:

Error The request was already sent.

```
1 function verifyHost()
```

Set to false to disable verification of the server's hostname in the server-provided TLS/x509 certificate. Defaults to true (i.e. verify). Only applicable to [https](#) URLs.

Throws:

Error The request was already sent.

```
1 function verifyCertificateTime(bool)
```

Set to false to disable verification of the server's TLS/x509 certificate validity time. Defaults to true (i.e. verify). Only applicable to [https](#) URLs.

Throws:

Error The request was already sent.

```
1 function onCompleted(callback, response_type)
```

`response_type` must be either [string](#) or [json](#) and determines how the response payload should be parsed and provided.

The callback should have the signature `function(status, response, payload)`. Status will either be [success](#) or [error](#). In case [json](#) was set as the response type, an additional [parse error](#) is possible if the payload could not be parsed as JSON. `response` will be an instance of `HttpResponse`. The payload is either a string or an JSON object if `response_type` was specified as [json](#). In case of a JSON parse error, it contains the parse failure as a string.

Throws:

Error The request was already sent.

TypeError callback not callable or invalid `response_type`

```
1 function execute()
```

Sends a previously prepared request. No further modifications are possible. The callback provided to `onCompleted()` will be invoked on completion.

15.7.2.2. FormData

Encapsulates `multipart/form-data` to be sent via the HTTP POST request payload.

```
1 function addContent(name, data, contentType)
```

Adds an element to the FormData instance with the given name. `data` must either be a string or a `Buffer` instance. `contentType` is optional and will default to `application/octet-stream` if unspecified.

```
1 function addFileContent(name, filename, data, contentType)
```

Adds a file element to the FormData instance with the given name. `data` must either be a string or a `Buffer` instance. `contentType` is optional and will default to `application/octet-stream` if unspecified.

15.7.3. HttpResponse

```
1 function status()
```

Obtains the HTTP response status code.

```
1 function header(name)
```

Returns the HTTP response header with the specified name or `undefined` if no such header exists.

15.8. HTTP Server API

```
1 var HttpServer = require('sys/httpserver');
```

15.8.1. Exported functions

Allows registering a handler to let a script server HTTP requests:

```
1 function registerHandler(path, callback, options)
```

Parameters:

- path** The URL suffix to register the handler for
- callback** The function to be invoked to handle a request. The function will be invoked with the `HttpRequest` instance as the parameter.
- options** Optional handler options

The handler will be accessible via `Http` with the prefix `/ws/`, e.g. `http://hostname/ws/path`.



By default, no access control is enforced.

If provided, **options** must be given as an object.

Defined options:

require_scope Require a bearer token for authentication with the given scope. Introduced in firmware version 4.1.

15.8.2. Classes

15.8.2.1. HttpRequest

Represents an incoming Http request to be handled.

```
1 function requestMethod()
```

Returns the Http request method as a string, e.g. **GET** or **POST**.

```
1 function requestHeader(header)
```

Allows to obtain an Http request header, e.g. an **Authorization** header. Returns the header value as a string or undefined if the header is not present.

```
1 function urlParam(param)
```

Allows to obtain a parameter encoded in the request URL, i.e. via an **?key=value** addition to the url. Returns the parameter as string or undefined if the parameter is not present.

```
1 function postParam(param)
```

Allows to obtain a parameter for a POST request where the POST payload is encoded as **application/x-www-form-urlencoded**. Returns the parameter as string or undefined if the parameter is not present.

If the payload is encoded as **application/json** and is an object, returns the mapped value for the given key. Return type is the type of the mapped value in this case.

```
1 function postParams()
```

Returns the POST payload.

In case of an **application/x-www-form-urlencoded** payload, this is an object with the received key-value pairs.

In case of an **application/json** payload, this is the payload as received.

```
1 function status(statusCode)
```

Sets the Http status code for the response. May only be called as long as neither `write()` nor `complete()` was invoked. If not invoked, a status of `200 OK` will be sent.

Throws:

Error Response headers were already sent, i.e. `write()` or `complete()` was invoked.

```
1 function header(header, value)
```

Sets a header on the response. The header name passed must be the bare header name, i.e. not contain a trailing colon. May only be called as long as neither `write()` nor `complete()` was invoked.

Throws:

Error Response headers were already sent, i.e. `write()` or `complete()` was invoked.

```
1 function write(payload)
```

Writes the response payload. May be invoked multiple times, but only as long as `complete()` was not invoked. After invoking `write()`, no status code or response header may be set anymore.

Throws:

Error Request was already completed via `complete()` and sent.

```
1 function complete(payload)
```

Indicates that the response was prepared and is to be sent. No further methods can be invoked on this instance.

Throws:

Error Request was already completed via `complete()` and sent.

16. Examples

16.1. Toggle a light every 5s seconds

The following is a minimal example to toggle a light (On/Off cluster) every 5 seconds. The timers and zigbee modules are imported as `Timer` and `ZigBee`. A callback function is registered via `ZigBee.onReady()` to be invoked once the ZigBee subsystem is initialized and devices can be looked-up. The light device is looked-up by its address and the On/Off cluster on application #10 is obtained. Finally, a periodic timer is created to execute the `toggle()` command on the On/Off cluster and the obtained status is printed into the log.

```
1 var Timer = require('sys/timers');
2 var ZigBee = require('sys/zigbee');
3
4 ZigBee.onReady(function()
5 {
6     // Get the light
7     var device = ZigBee.getDevice('00:21:2E:FF:FF:00:58:F1');
8     var onOff = device.getApplication(10).getOnOffCluster();
9
10    Timer.createPeriodicTimer('5s', function()
11    {
12        onOff.toggle(function(status)
13        {
14            print('toggle:', status);
15        });
16    });
17 });
```

16.2. Control a light via a motion detector

The following example demonstrates how attributes can be evaluated to trigger further actions. The “Occupancy” attribute of an occupancy sensor (motion detector) is evaluated and a light is switched on whenever a motion is reported and switched off once no motion is detected anymore.

```

1 var ZigBee = require('sys/zigbee');
2
3 // Will keep a reference to the kitchen light (the application instance)
4 var kitchenLight;
5
6 // called whenever the Occupancy attribute changes
7 function onOccupancyStateChange(attributeID, attribute) {
8     print('motion sensor value changed to', attribute);
9
10    if (attribute.testBit(0)) {
11        // Occupied, set level 192 (75%) within 1s (10 units of 0.1s each)
12        kitchenLight.getLevelControlCluster().moveToLevelWithOnOff(192, 10,
13            function (status) {
14                print('kitchen light turned on:', status)
15            });
16    } else {
17        kitchenLight.getOnOffCluster().off(function (status) {
18            print('kitchen light turned off:', status)
19        });
20    }
21 }
22
23 function setupKitchenLighting() {
24     // Get the kitchen light and store it in the global variable kitchenLight
25     kitchenLight = ZigBee.getDevice('00:1F:EE:00:00:00:01:A3').
26         getApplication(1);
27
28     // Get the motion sensor
29     var MotionSensor = ZigBee.getDevice('00:0D:6F:00:04:B1:0B:1F');
30     var occupancy = MotionSensor.getApplication(1).getCluster(0x0406);
31
32     // Register for attribute changes on attribute 0 (Occupancy, bitmap)
33     occupancy.onAttributeChanged(0x0000, onOccupancyStateChange);
34 }
35
36 ZigBee.onReady(function () {
37     setupKitchenLighting();
38 });

```

16.3. Write the StartUpOnOff attribute in the On/Off cluster

```

1 var zigbee = require('sys/zigbee');
2
3 zigbee.onReady(function() {
4     var endpoint = zigbee.lookup("00:1f:ee:00:00:70:95:09/1")
5     var onoff = endpoint.getOnOffCluster();
6
7     onoff.writeAttribute(0x4003, "enum8", 0,
8         function(status) {
9             print("Write StartUpOnOff:", status);
10        });
11 });

```

16.4. Control a light via an HTTP handler

```
1 var httpserver = require('sys/httpserver')
2 var zigbee = require('sys/zigbee')
3
4 var onoff;
5
6 zigbee.onReady(function() {
7     onoff = zigbee.lookup("00:1f:ee:00:00:00:95:09/1").getOnOffCluster();
8
9     httpserver.registerHandler("/toggle",
10         function(request) {
11             // Handle http request: toggle the light
12             onoff.toggle(
13                 function(status)
14                 {
15                     if (status.success())
16                     {
17                         // On success, return HTTP status 200 OK
18                         request.status(200);
19                     }
20                     else
21                     {
22                         // On failure, return status 502 Bad Gateway and
23                         // provide the error message.
24                         request.status(502);
25                         request.write(status.toString());
26                     }
27
28                     // Complete the HTTP request and send the response
29                     request.complete();
30                 });
31         });
32 });
```

16.5. Send an HTTP request on button press

```

1 var zigbee = require('sys/zigbee')
2 var http = require('sys/http');
3
4 function onToggle() {
5     // Send a POST request to http://10.0.8.10:8080/test with some payload and a
6     // bearer token for authentication
7     var request = http.request();
8     request.method("POST");
9     request.url("http://10.0.8.10:8080/test");
10    request.header("Authentication", "bearer 12345678901234567890");
11    request.postData({"key1": "value1", "test": 1234});
12    request.onCompleted(
13        // Invoked on completion of the request
14        function(status, response, payload)
15        {
16            print(status);
17            print(response);
18            print(payload);
19        }, "string");
20    // Request setup, now execute it
21    request.execute();
22 }
23
24 zigbee.onReady(function() {
25     var app = zigbee.lookup("01:52:00:45/1");
26     var onoff = app.getOnOffClientCluster();
27
28     // Invoke function onToggle() whenever a Toggle command on the On/Off
29     // cluster is received.
30     onoff.on('off', onToggle);
31 })

```

17. Revision History

Revision	Date	Remarks
0.1	08/24/2021	Initial draft
0.2	08/26/2021	Added description in automations overview, bundle structure, bundle.info, configuration.schema
0.3	09/02/2021	Updated automations overview, bundle structure, bundle.info, configuration.schema
0.4	09/03/2021	Updated configuration.schema, added information for zone level settings and filters
0.41	09/03/2021	Updated configuration.schema, added information for global/zone level settings
0.42	09/09/2021	Added attribute information in zone level settings
0.43	09/10/2021	Added/updated attribute information in zone level settings
0.44	09/10/2021	Updated overall structure and made the document more generic
0.45	09/13/2021	Added examples for each attribute
0.46	09/13/2021	Added examples for some attributes and customized examples
0.47	09/14/2021	Document restructured, added zone/global level settings information
0.48	09/14/2021	Updated the main example configuration.schema
0.49	09/16/2021	Updated the document structure, added initial description
0.50	09/17/2021	Added supported icons and the reference to the JavaScript documentation
0.51	09/20/2021	Added supported icons images from iOS app
2.0	02/02/2022	Added JavaScripts runtime part
2.1	02/22/2023	Added JavaScript documentation for globals and Http client/server
2.2	05/05/2023	Updated JavaScript documentation to G1 firmware v4.1
2.3	09/04/2023	Updated JavaScript documentation to G1 firmware v4.1.6

18. Contact

ubisys technologies GmbH
Neumannstr. 10
40235 Düsseldorf
Germany

T: +49. 211. 54 21 55 - 19
F: +49. 211. 54 21 55 - 99

www.ubisys.de
info@ubisys.de