



# **RAPIX Logic Programming Guide**

**Version 15, 1 June 2021**

|       |                              |    |
|-------|------------------------------|----|
| 1     | Contents                     |    |
| 2     | Document History .....       | 5  |
| 3     | Introduction .....           | 6  |
| 4     | How it Works.....            | 7  |
| 5     | Creating Logic.....          | 8  |
| 5.1   | Adding Logic Modules .....   | 8  |
| 5.2   | Module Groups .....          | 9  |
| 5.3   | Adding Global Code.....      | 10 |
| 5.4   | Writing Logic Code .....     | 10 |
| 5.4.1 | Code Comments.....           | 10 |
| 5.4.2 | Keyboard Short-cuts.....     | 10 |
| 5.4.3 | Code Snippet Generator ..... | 11 |
| 5.5   | Compiling .....              | 11 |
| 5.6   | Transferring.....            | 11 |
| 6     | Language Reference.....      | 11 |
| 6.1   | General.....                 | 11 |
| 6.2   | Module Environment .....     | 12 |
| 6.3   | RAPIX Entities.....          | 12 |
| 6.3.1 | Zones .....                  | 12 |
| 6.3.2 | Scenes .....                 | 16 |
| 6.3.3 | Xi Flags.....                | 17 |
| 6.3.4 | Xi Operating Properties..... | 19 |
| 6.4   | RAPIX Extensions.....        | 21 |
| 6.5   | Delays.....                  | 21 |
| 6.5.1 | How it works .....           | 22 |
| 6.5.2 | Limitations.....             | 22 |
| 6.6   | Timers.....                  | 23 |
| 6.6.1 | Properties.....              | 23 |
| 6.6.2 | Methods.....                 | 23 |
| 6.6.3 | Usage.....                   | 23 |
| 6.6.4 | Examples .....               | 24 |
| 6.7   | UDP Client .....             | 25 |
| 6.7.1 | Properties.....              | 25 |
| 6.7.2 | Methods.....                 | 25 |
| 6.7.3 | Example.....                 | 25 |
| 6.8   | TCP/IP Client .....          | 27 |

|        |                                    |    |
|--------|------------------------------------|----|
| 6.8.1  | Properties.....                    | 27 |
| 6.8.2  | Methods.....                       | 27 |
| 6.8.3  | Example.....                       | 27 |
| 6.9    | Serial Port.....                   | 28 |
| 6.9.1  | Properties.....                    | 28 |
| 6.9.2  | Methods.....                       | 28 |
| 6.9.3  | Constants .....                    | 28 |
| 6.9.4  | Constructor .....                  | 29 |
| 6.9.5  | Example.....                       | 29 |
| 6.10   | Other Properties and Methods.....  | 31 |
| 6.10.1 | Date and Time .....                | 31 |
| 6.10.2 | Connectivity .....                 | 31 |
| 6.10.3 | Logic Module.....                  | 31 |
| 6.10.4 | Level Conversion .....             | 31 |
| 6.10.5 | Other .....                        | 32 |
| 6.11   | Constants .....                    | 32 |
| 7      | Best-Practices.....                | 33 |
| 7.1    | Comments and naming.....           | 33 |
| 7.2    | Module Period .....                | 33 |
| 7.3    | Performing Actions on Changes..... | 33 |
| 8      | Debugging .....                    | 34 |
| 9      | Examples .....                     | 35 |
| 9.1    | Zone Toggle.....                   | 35 |
| 9.2    | Zone Level Adjustment .....        | 35 |
| 9.3    | Zone Tracking.....                 | 36 |
| 9.4    | Conditional Zone Tracking .....    | 36 |
| 9.5    | Logical OR.....                    | 36 |
| 9.6    | Logical AND .....                  | 36 |
| 9.7    | Logical NOT .....                  | 36 |
| 9.8    | Adjacent Area Control.....         | 37 |
| 9.9    | Initialisation.....                | 38 |
| 9.9.1  | Global Declaration .....           | 38 |
| 9.9.2  | Run Once Module .....              | 38 |
| 9.9.3  | Using IsFirstRun.....              | 38 |
| 9.10   | Using Date and Time .....          | 39 |
| 9.10.1 | Time of Day .....                  | 39 |

|        |  |    |
|--------|--|----|
| 9.10.2 | Day of Week .....                        | 39 |
| 9.10.3 | Day of Year .....                        | 39 |
| 9.10.4 | Sunrise/sunset .....                     | 39 |
| 9.11   | Enabling / Disabling Logic Modules ..... | 40 |
| 9.11.1 | Excluded Modules .....                   | 40 |
| 9.11.2 | Enable / Disable methods .....           | 40 |
| 9.11.3 | Conditional Operation.....               | 40 |
| 9.12   | Sequence of Events .....                 | 42 |
| 9.13   | Handling Error Conditions .....          | 44 |

## 2 Document History

| Version | Date      | Changes  |
|---------|-----------|--|
| 12      | 26/2/2021 | Initial release  |
| 13      | 16/4/2021 | Added document history<br>Added serial port methods<br>Added Level Conversion methods<br>Added examples about code initialisation<br>Renamed top-level item in module tree |
| 14      | 5/5/2021  | Fixed typos in serial port section.  |
| 15      | 1/6/2021  | Clarified Zone state property.<br>Added examples.  |

### 3 Introduction

The templates used in RAPIX Xi Devices provide the ability to customise the behaviour of a RAPIX system. Sometimes, however, there are requirements that are beyond the scope of these templates, and this is where the RAPIX Logic is useful.

RAPIX Logic provides a general-purpose scripting environment that is customised to the needs of building automation. It is based on one of the most popular programming languages, C#.

The C# language is not discussed in this document. There are myriad sources on-line that can be used for learning how to program with C#. A good starting point for learning C# is <https://www.w3schools.com/cs/>. Only a very small sub-set of the C# language will typically be required for RAPIX Logic.

**RAPIX Logic should only be used by people with  
experience with the RAPIX Lighting Control  
System and with C# programming.**

**Use at your own risk.**

## 4 How it Works

The user can create one or more "modules" of logic. Each module is executed at a rate selected by the user, from several times per second, up to every hour or more. The modules are completely independent of each other.

The Logic code has access to various aspects of the RAPIX system that can be used to make decisions. These include:

- Zone levels, colours, error conditions and occupancy status
- Scene state
- Xi Flag state
- Xi Operating Property values
- System status
  - Date/time, sunrise and sunset time
  - Ethernet connected
  - Modbus connected

The Logic code also has access to various aspects of the RAPIX system that can be controlled. These include:

- Zone levels and colours
- Scene state
- Xi Flag state
- Xi Operating Property values
- System functions
  - UDP sockets
  - Logging

There are two types of logic code:

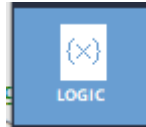
- Modules: These contain the logic code which is executed periodically.
- Global code: This provides a mechanism for defining global (shared) variables and methods which are used by the modules. Most basic logic code will not require global code.

The user generated Logic Code is compiled and transferred to the Zone Controller. The Master Zone controller executes the modules at a rate selected by the user.

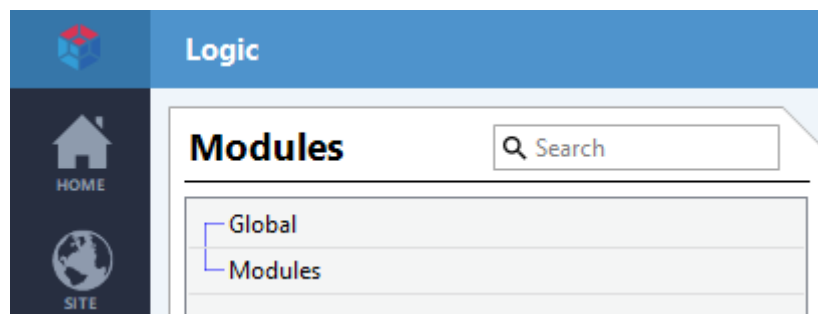
## 5 Creating Logic

### 5.1 Adding Logic Modules

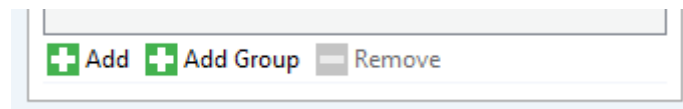
Select the logic tab from the left-panel:



The tree shows the user's Logic code, sorted into Global code and Modules:

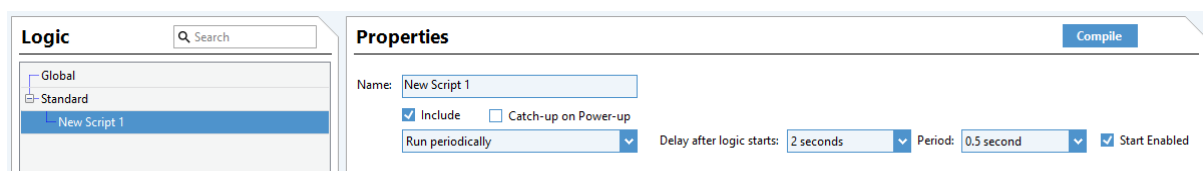


To add Logic code, select the Modules node, and click on the Add button:



Select the new module in the tree and select its properties:

- Name
- When to start running it (Delay after logic starts)
- How often to run it (Run Period)
- Whether the logic should be initially enabled (this can be changed after the logic starts if needed)



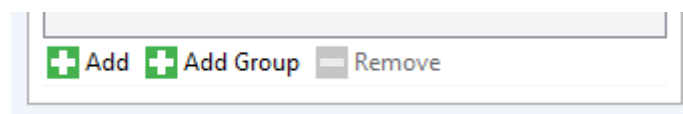
There are options for when to run the logic module:

| Option                  | Description  |
|-------------------------|--|
| Don't run automatically | The module will not run until it has been requested. Once enabled (by another module), it will run once.                             |
| Run once                | The module will run once at a selected time after the logic starts.  |
| Run periodically        | The module will run at a selected time after the logic starts, then at a selected period (interval). This is the "normal" operation. |

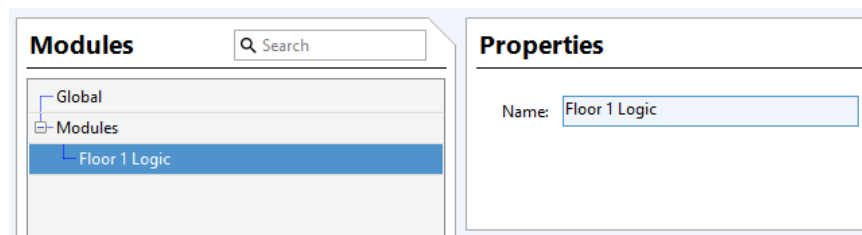
## 5.2 Module Groups

Modules can be put into groups for convenience.

To add a Logic Group, select the Global or Modules node (or another Group), and click on the Add Group button:

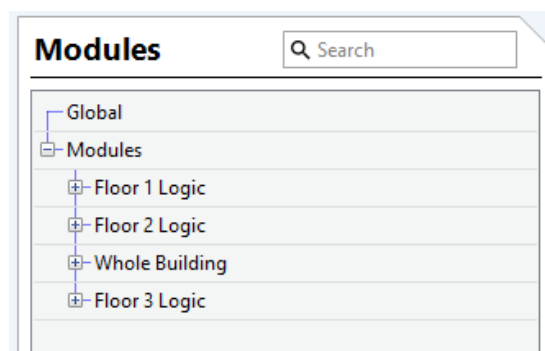


Select the new group in the tree and give it a name:



Logic Modules can be added under a group by selecting the group node, then clicking on the Add button.

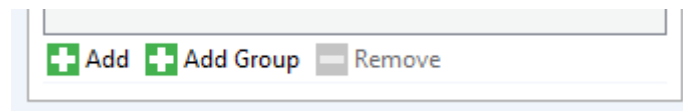
The result can be a hierarchy of any structure to suit the needs of the project.



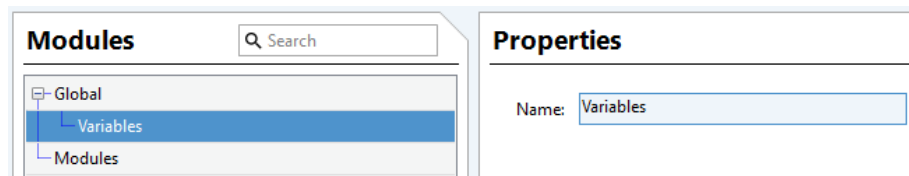
***Example of Logic Grouping***

### 5.3 Adding Global Code

Global code is only required when implementing more complex features. To add global code, select the Global node, and click on the Add button:

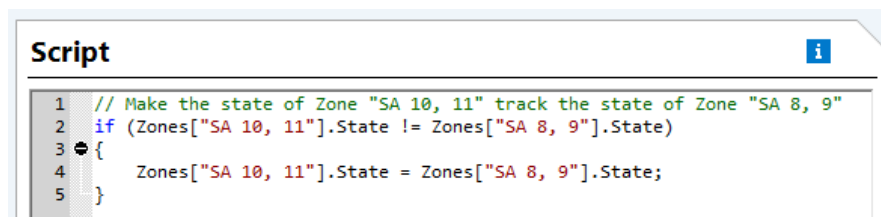


Give the new global code item a name that describes its purpose:



### 5.4 Writing Logic Code

Logic Code can be typed directly into the Script editor. The code will be syntax highlighted as it is entered:



#### 5.4.1 Code Comments

It is always good practice to put as many comments in code as possible. Comments at the start of a module will be used as a description of the module in the project summary. All comments up until the first line that is not a comment (i.e. is a blank line or code) will be included.

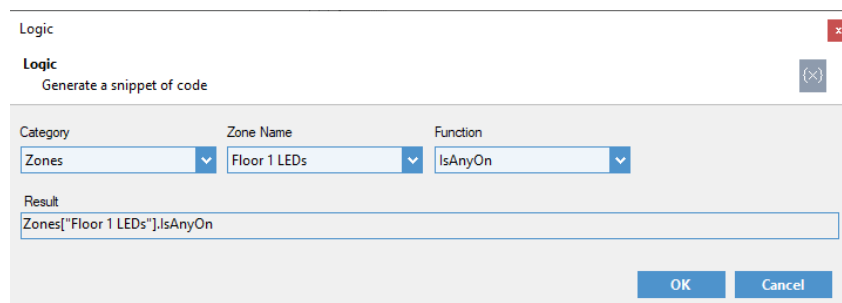
#### 5.4.2 Keyboard Short-cuts

Keyboard short-cuts that can be used in the logic editor are shown in the table below.

| Key Combination | Function                                |
|-----------------|---|
| CTRL + SPACE    | Show code snippet generator (see below) |
| CTRL + A        | Select All text                         |
| CTRL + C        | Copy selected text                      |
| CTRL + D        | Duplicate current line                  |
| CTRL + L        | Delete current line                     |
| CTRL + T        | Toggle (swap) line with the one above   |
| CTRL + V        | Paste                                   |
| CTRL + X        | Cut                                     |
| CTRL + Z        | Undo                                    |

### 5.4.3 Code Snippet Generator

For quickly inserting code "snippets", select the position in the code editor where it is to be entered and click on CTRL + SPACE. The Logic snippet editor will appear:

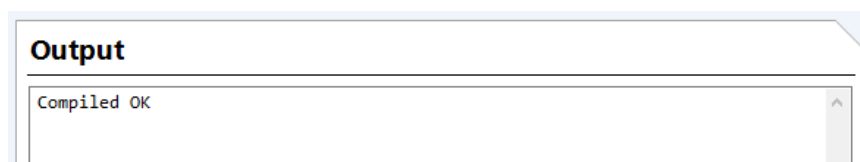


The screenshot shows a dialog box titled "Logic" with a close button (X) in the top right corner. Below the title bar, there is a "Logic" label and a "Generate a snippet of code" button with a code icon. The main area contains three dropdown menus: "Category" (set to "Zones"), "Zone Name" (set to "Floor 1 LEDs"), and "Function" (set to "IsAnyOn"). Below these is a "Result" text box containing the code snippet: `Zones["Floor 1 LEDs"].IsAnyOn`. At the bottom right are "OK" and "Cancel" buttons.

Select the code category and other options as required. The resulting code will be displayed. When finished, click on OK. The code will be inserted in the code editor.

## 5.5 Compiling

When the Logic is complete, click on the Compile button. If all is correct, the message "Compiled OK" will be seen in the output window:



Any errors will be displayed in the Output window. You will need to check the listed module and line number to determine the cause. ***Please note that the reported error position within a line may not be correct due to the pre-processing of the logic code.***

## 5.6 Transferring

Once the Logic has been successfully compiled, transfer the project to the Zone Controllers. The Zone Controllers will re-boot to load the logic. The logic will start running once the DALI Line scans are complete.

# 6 Language Reference

## 6.1 General

The RAPIX Logic uses the .Net 3.5 framework. More recent language features are not supported.

The RAPIX Logic code is run in a "sand-boxed" environment for security. A result of this is that many standard name-spaces are not available.

## 6.2 Module Environment

The code in the logic Modules or Global code can access anything in the Global code.

The code in the logic Modules can use the RAPIX "context", which is a structure named "Rapix". This provides the interface to the RAPIX Zones and other entities.

Note that the logic code is pre-processed to insert the class name "Rapix" where needed. This eliminates the need for a lot of text. The following are equivalent:

- `Rapix.Zones[3]`
- `Zones[3]`

## 6.3 RAPIX Entities

### 6.3.1 Zones

#### 6.3.1.1 Zone Class

The "Zone" class provides access to the RAPIX Zones. There should be no need to create these objects directly. The context will provide them for the Logic to use through the `Rapix.Zones` dictionary (see below).

##### 6.3.1.1.1 Properties

The Zone class properties are shown below. The less commonly used properties are shown in grey.

| Property Name           | Type   | Settable | Description  |
|-------------------------|--------|----------|--|
| AverageLevel            | int    | No       | The average level in the zone                                    |
| Error                   | bool   | No       | Whether there is an error in the zone                            |
| FadeTime                | float  | Yes      | The zone fade time (in seconds)                                  |
| IsAllOff                | bool   | No       | Whether everything in the zone is off                            |
| IsAllOn                 | bool   | No       | Whether everything in the zone is on                             |
| IsAnyOff                | bool   | No       | Whether anything in the zone is off                              |
| IsAnyOn                 | bool   | No       | Whether anything in the zone is on                               |
| IsValidData             | bool   | No       | Whether the zone information is valid                            |
| IsFading                | bool   | No       | Whether the zone is fading                                       |
| IsOccupied              | bool   | No       | Whether the zone is occupied (i.e. a sensor has detected motion) |
| IsOff                   | bool   | No       | Whether the zone is off (same as IsAllOff)                       |
| IsOn                    | bool   | No       | Whether the zone is on (same as IsAnyOn)                         |
| Level                   | int    | Yes      | The zone level (see below)                                       |
| LevelsAreAllEqual       | bool   | No       | Whether the level of everything in the zone is the same          |
| MinLevel                | int    | No       | The minimum level in the zone                                    |
| MaxLevel                | int    | No       | The maximum level in the zone                                    |
| Name                    | string | No       | The name   |
| Number                  | int    | No       | The zone number (id)   |
| State                   | bool   | Yes      | Whether the zone is on (same as IsAnyOn)                         |
| Tag                     | int    | Yes      | A value that can be used for any purpose                         |
| TargetLevel             | int    | No       | The level that the zone is fading towards (if fading)            |
| TargetLevelsAreAllEqual | bool   | No       | Whether the target level of everything in the zone is the same   |

Zone levels use the DALI level, 0 – 254. They can also be referred to in %, which will be converted to a DALI Level during the pre-process stage. For example, the following are equivalent:

- `Zones["Office 1"].Level = 50%;`
- `Zones["Office 1"].Level = 127;`

Reading the `Zone.Level` property value gives the same value as the `TargetLevel`. This is:

- The maximum level in the Zone if the Zone is not fading; or
- The maximum target level in the Zone if the Zone level is fading.

Setting the `Zone.Level` property is the same as `Zone.FadeToLevel()` with a fade time of 0.

The `Zone.AverageLevel` property can be used to get an indication of the "level" of a Zone, but it may not give the expected value if:

- Devices have non-standard min/max settings; or
- Devices are fading from one level to another; or
- There are relay devices in the Zone; or
- Different parts of the Zone have been set to different levels.

For this reason, **it is recommended that `Zone.Level` be used for most purposes.**

#### 6.3.1.1.2 Zone Colours

The properties related to Zone colour are:

| Property Name     | Type | Settable | Description                                    |
|-------------------|------|----------|--|
| Color             | long | Yes      | The Zone colour                                |
| ColorRGB          | int  | No       | The Zone colour in RGB                         |
| ColorRGBWAF       | long | No       | The Zone colour in RGBWAF                      |
| ColorTemp         | int  | No       | The Zone colour temperature (in Kelvin)        |
| ColorType         | enum | No       | The Zone colour type                           |
| ColorXY           | long | No       | The Zone colour in XY                          |
| HasColor          | bool | No       | Whether the zone has colour devices            |
| TargetColor       | long | No       | The Zone target colour                         |
| TargetColorRGB    | int  | No       | The Zone target colour in RGB                  |
| TargetColorRGBWAF | long | No       | The Zone target colour in RGBWAF               |
| TargetColorTemp   | int  | No       | The Zone target colour temperature (in Kelvin) |
| TargetColorXY     | long | No       | The Zone target colour in XY                   |

Zone Colours are long int values as shown in the table below.

| Colour Type | Byte |   |     |       |       |       |       |       |
|-------------|------|---|-----|-------|-------|-------|-------|-------|
|             | 7    | 6 | 5   | 4     | 3     | 2     | 1     | 0     |
| Colour Temp | -    | 1 | -   | -     | -     | -     | MSB   | LSB   |
| RGBWAF      | -    | 3 | Red | Green | Blue  | White | Amber | Free  |
| XY          | -    | 4 | -   | -     | X MSB | X LSB | Y MSB | Y LSB |

Colour Temperature is in Kelvin.

RGB values are an int value as shown below:

| Colour Type | Byte |     |       |      |
|-------------|------|-----|-------|------|
|             | 3    | 2   | 1     | 0    |
| RGB         | -    | Red | Green | Blue |

All colour components have values in the range 0 to 254. The value 255 (0xFF) is "MASK", meaning no change.

Examples:

- Colour Temperature, 3000K = 0x0001000000000BB8
- Colour Temperature, 5000K = 0x0001000000001388
- Colour Temperature, Mask = 0x000100000000FFFF
- RGB(WAF) Red = 0x0003FE0000000000
- RGB(WAF) White = 0x0003FEFEFEFE0000
- RGB(WAF) MASK = 0x0003FFFFFFFFFFFF

The Zone colour type is an enumeration:

```
public enum DaliColorType
{
    ColorTemperature = 1,
    RGB = 3,
    XY = 4,
    None = 0xFF
}
```

#### 6.3.1.1.3 Methods

The Zone class methods are:

| Method                                 | Description                    |
|--|--------------------------------|
| FadeToLevel(int level, float fadeTime) | Fade the zone to the new level |
| Off()                                  | Turn the zone off              |
| On()                                   | Turn the zone on               |
| StopFade()                             | Stop the zone fading           |

The methods related to Zone colour are:

| Method   | Description                             |
|--|---|
| FadeToColor(long color, float fadeTime)                          | Fade the zone to a new colour           |
| FadeToColorRGB(int color, float fadeTime)                        | Fade the zone to a new colour           |
| FadeToColorRGBWAF(long color, float fadeTime)                    | Fade the zone to a new colour           |
| FadeToColorTemp(int color, float fadeTime)                       | Fade the zone to a new colour           |
| FadeToColorXY(int color, float fadeTime)                         | Fade the zone to a new colour           |
| FadeToLevelAndColor(int level, long color, float fadeTime)       | Fade the zone to a new level and colour |
| FadeToLevelAndColorRGB(int level, int color, float fadeTime)     | Fade the zone to a new level and colour |
| FadeToLevelAndColorRGBWAF(int level, long color, float fadeTime) | Fade the zone to a new level and colour |
| FadeToLevelAndColorTemp(int level, int color, float fadeTime)    | Fade the zone to a new level and colour |
| FadeToLevelAndColorXY(int level, int color, float fadeTime)      | Fade the zone to a new level and colour |

#### 6.3.1.2 Zones Dictionary

The RAPIX Zones are accessible from the Rapix.Zones dictionaries.

The Zones can be indexed (looked-up) by id or name.

#### 6.3.1.3 Examples

To perform an action if the level of a Zone called "Office 1" is more than 50%:

```
if (Zones["Office 1"].Level > 50%)
{
    ...
}
```

To turn on a Zone called "Office 1":

```
Zones["Office 1"].On();
```

To fade a Zone called "Lobby" to colour temperature 3000K (without changing the level) over 30 seconds:

```
Zones["Lobby"].FadeToColorTemp(3000, 30);
```

To fade a Zone called "Lobby" to level 100%, blue (RGB #0000FE) over 1 second:

```
Zones["Lobby"].FadeToLevelAndColorRGB(100%, 0x0000FE, 1);
```

## 6.3.2 Scenes

### 6.3.2.1 Scene Class

The "Scene" class provides access to the RAPIX Scenes. There should be no need to create these objects directly. The context will provide them for the Logic to use through the Rapix.Scenes dictionary (see below).

#### 6.3.2.1.1 Properties

The Scene class properties are:

| Property Name | Type   | Settable | Description                              |
|---------------|--------|----------|--|
| IsSet         | bool   | No       | Whether the scene is set                 |
| Name          | string | No       | The name                                 |
| Number        | int    | No       | The scene number (id)                    |
| Tag           | long   | Yes      | A value that can be used for any purpose |

#### 6.3.2.1.2 Methods

The Scene class methods are:

| Method | Description        |
|--------|--------------------|
| Off()  | Turn the scene off |
| Set()  | Set the scene      |

### 6.3.2.2 Scenes Dictionary

The RAPIX Scenes are accessible from the Rapix.Scenes dictionaries.

The Scenes can be indexed (looked-up) by id or name.

### 6.3.2.3 Examples

To perform an action if the level of a Scene called "Morning" is set:

```
if (Scenes["Morning"].IsSet)
{
    ...
}
```

To set a scene called "Morning":

```
Scenes["Morning"].Set();
```

### 6.3.3 Xi Flags

#### 6.3.3.1 *XiFlagGroup Class*

The "XiFlagGroup" class provides access to the RAPIX Xi Flag Groups. There should be no need to create these objects directly. The context will provide them for the Logic to use through the Rapix.XiFlagGroups dictionary (see below).

##### 6.3.3.1.1 Properties

The XiFlagGroup class properties are:

| Property Name   | Type                    | Settable | Description            |
|-----------------|-------------------------|----------|------------------------|
| FlagGroupNumber | Int                     | No       | The Flag Group number  |
| Flags           | Dictionary<int, XiFlag> | No       | The flags in the group |
| Name            | string                  | No       | The name               |

##### 6.3.3.1.2 Methods

There are no XiFlagGroup methods.

#### 6.3.3.2 *XiFlagGroups Dictionary*

The RAPIX Zones are accessible from the Rapix.FlagGroups dictionaries.

The Xi Flag Groups can be indexed (looked-up) by id or name.

#### 6.3.3.3 *XiFlag Class*

The "XiFlag" class provides access to the RAPIX Xi Flags. There should be no need to create these objects directly. The context will provide them for the Logic to use through the XiFlagGroup.Flags dictionary (see below).

##### 6.3.3.3.1 Properties

The XiFlag class properties are:

| Property Name | Type        | Settable | Description                              |
|---------------|-------------|----------|--|
| FlagGroup     | XiFlagGroup | No       | The Flag Group                           |
| FlagNumber    | int         | No       | The flag number                          |
| IsClear       | bool        | Yes      | Whether the flag is clear (i.e. not set) |
| IsSet         | bool        | Yes      | Whether the flag is set (same as State)  |
| Name          | string      | No       | The name                                 |
| State         | bool        | Yes      | Whether the flag is set                  |
| Tag           | long        | Yes      | A value that can be used for any purpose |

#### 6.3.3.3.2 Methods

The XiFlag methods are:

| Method  | Description    |
|---------|----------------|
| Clear() | Clear the flag |
| Set()   | Set the flag   |

#### 6.3.3.4 Flags Dictionary

The RAPIX Xi Flags are accessible from the XiFlagGroup.Flags dictionaries.

The Xi Flags can be indexed (looked-up) by id or name.

#### 6.3.3.5 Examples

To perform an action if Flag Group "Sensors", Flag "Meeting Room" is set:

```
if (FlagGroups["Sensors"].Flags["Meeting Room"].State)
{
    ...
}
```

To set Flag Group "Sensors", Flag "Meeting Room":

```
FlagGroups["Sensors"].Flags["Meeting Room"].Set();
```

The following are all equivalent for checking that a Flag is set:

```
if (FlagGroups[1].Flags[2].State)...
if (FlagGroups[1].Flags[2].State == SET)...
if (FlagGroups[1].Flags[2].IsSet)...
```

The following are all equivalent for checking that a Flag is clear:

```
if (!FlagGroups[1].Flags[2].State)...
if (FlagGroups[1].Flags[2].State == CLEAR)...
if (FlagGroups[1].Flags[2].IsClear)...
```

### 6.3.4 Xi Operating Properties

#### 6.3.4.1 *XiOpProp Class*

The "XiOpProp" class provides access to the RAPIX Xi Operating Properties. There should be no need to create these objects directly. The context will provide them for the Logic to use through the Rapix.OpProps dictionary (see below).

##### 6.3.4.1.1 Properties

The XiOpProp class properties are:

| Property Name | Type                  | Settable | Description                              |
|---------------|-----------------------|----------|--|
| Name          | string                | No       | The name                                 |
| OpPropNumber  | int                   | No       | The Operating property number            |
| Value         | int                   | Yes      | The current value                        |
| Values        | List<XiOpPropValue>() | No       | A list of named values                   |
| Tag           | long                  | Yes      | A value that can be used for any purpose |

##### 6.3.4.1.2 Methods

The XiOpProp class methods are:

| Method                     | Description   |
|----------------------------|---|
| bool IsValue(int number)   | Whether the Operating Property is this value number |
| bool IsValue(string name)  | Whether the Operating Property is this value name   |
| void SetValue(int number)  | Set the Operating Property to this value number     |
| void SetValue(string name) | Set the Operating Property to this value name       |

#### 6.3.4.2 *XiOpProps Dictionary*

The RAPIX Xi Operating Properties are accessible from the Rapix.OpProps dictionaries.

The Xi Operating Properties can be indexed (looked-up) by id or name.

#### 6.3.4.3 *XiOpPropValue Class*

The "XiOpPropValue" class provides access to the RAPIX Xi Operating Property. There should be no need to create these objects directly. The context will provide them for the Logic to use through the XiOpProps.Values dictionary (see below).

**Note: it is rarely necessary to use the XiOpPropValue objects.**

##### 6.3.4.3.1 Properties

The XiOpPropValue class properties are:

| Property Name | Type     | Settable | Description                   |
|---------------|----------|----------|-------------------------------|
| Name          | string   | No       | The name                      |
| OpProp        | XiOpProp | No       | The parent operating property |
| Value         | int      | No       | The value                     |

#### 6.3.4.3.2 Methods

The XiOpPropValue class has no methods.

#### 6.3.4.4 XiOpPropValues Dictionary

The RAPIX Xi Operating Property Values are accessible from the XiOpProp.Values dictionaries.

The Xi Operating Property values can be indexed (looked-up) by id or name.

#### 6.3.4.5 Examples

To perform an action if the value of an Xi Operating property "Building" is "Work Hours":

```
if (OpProps["Building"].IsValue("Work Hours"))
{
    ...
}
```

To perform an action if the value of an Xi Operating property "Building" is **not** "After Hours":

```
if (!OpProps["Building"].IsValue("After Hours"))
{
    ...
}
```

To set the value of an Xi Operating property "Building" to "Work Hours":

```
OpProps["Building"].SetValue("Work Hours");
```

## 6.4 RAPIX Extensions

RAPIX Logic has a series of methods designed to simplify the common coding requirements for building automation.

The following methods are used to allow code to be executed only when something changes:

- HasChanged
- NowTrue
- NowFalse
- StayedTrue
- StayedFalse

To have some script execute only when a value changes, the code would be:

```
if (HasChanged(value))  
{  
    ...  
}
```

To have some script execute only when a value changes from false to true, the code would be:

```
if (NowTrue(value))  
{  
    ...  
}
```

To have some script execute only when a value has stayed true for a duration, the code would be:

```
if (StayedTrue(value, duration))  
{  
    ...  
}
```

## 6.5 Delays

Delays can be used in the code to pause between sections of the code. There are several methods provided for this:

- Delay
- DelayWhile
- DelayUntil

For a fixed delay of 5 seconds:

```
...  
Delay(5);  
...
```

To delay for an indeterminate period while a (Boolean) value or expression is true:

```
...
DelayWhile (value) ;
...
```

To delay for an indeterminate period until a value is true:

```
...
DelayUntil (value) ;
...
```

### 6.5.1 How it works

Delays are a convenient coding short-cut to avoid the need for timers, threads or asynchronous code.

When there is a delay in a module, the module is effectively split into two parts that are run separately. The process is essentially:

1. Run first part of module
2. Exit
3. Wait
4. Run next part of module

### 6.5.2 Limitations

When one or more delays are used, the two (or more) parts of the module are run completely independently of each other. This means that any state information from one part of the module is not available for later parts of the module.

For example, this will not work because the value of "level" is lost:

```
int level = 10;
Zones[3].Level = level;
Delay(2);    // Code exits here, then comes back later.
level++;     // Error - the value of level is no longer 10.
Zones[3].Level = level;
```

For the same reason it is not possible to use delays within code blocks:

- for, foreach, while or do loops
- if () {...}

Refer to the examples for alternatives to loops with delays.

**Delays cannot be used in global code.**

## 6.6 Timers

The `RapixTimer` class is useful for timing events, or for implementing delays. The methods and properties are shown in the tables below.

### 6.6.1 Properties

| Property Name | Type     | Settable | Description                                      |
|---------------|----------|----------|--|
| Running       | bool     | No       | True if the timer is running                     |
| Time          | int      | No       | How long (in seconds) the timer has been running |
| TimeFloat     | float    | No       | How long (in seconds) the timer has been running |
| TimeSpan      | TimeSpan | No       | How long the timer has been running              |

### 6.6.2 Methods

| Method                    | Description                           |
|---------------------------|---------------------------------------|
| <code>RapixTimer()</code> | Constructor                           |
| <code>Start()</code>      | Start (or re-start) the timer running |
| <code>Stop()</code>       | Stop the timer                        |

### 6.6.3 Usage

A timer needs to be declared in a global code so that it exists independently of the logic modules. It should be instantiated in the same place.

The `RapixTimer` object will maintain the correct duration even in the system clock is adjusted forward or back (for example at the start or end of Daylight Saving).

The timer value will roll-over after 24 days. It should only be used for timing periods less than this. For longer periods, use a `DateTime` structure to record the start time and subtract this from the current time:

```
DateTime period = Now.Subtract(start_time);
```

#### 6.6.4 Examples

The code below turns on a Zone 60 seconds after an Operating Property changes.

In global code:

```
RapixTimer EventTimer = new RapixTimer();
```

In module 1:

```
if (NowTrue(OpProps["Building Mode"].IsValue("Peak hours")))
{
    EventTimer.Start();
}
```

In module 2:

```
if (EventTimer.Time >= 60)
{
    EventTimer.Stop();
    Zones["Floor 1 / Line 1"].FadeToLevel(100%, 4);
}
```

## 6.7 UDP Client

### 6.7.1 Properties

The `UdpClient` class properties are:

| Property Name            | Type              | Settable | Description                      |
|--------------------------|-------------------|----------|----------------------------------|
| <code>IsConnected</code> | <code>bool</code> | No       | Whether the client has connected |

### 6.7.2 Methods

The `UdpClient` methods are:

| Method  | Description  |
|---|--|
| <code>Close()</code>                            | Close the connection   |
| <code>Connect(string hostName, int port)</code> | Connect to the host at the specified port                      |
| <code>byte[] ReadBytes()</code>                 | Read bytes from the socket                                     |
| <code>string ReadAsciiString()</code>           | Read ASCII encoded bytes from the socket and convert to string |
| <code>string ReadUtf8String()</code>            | Read UTF-8 encoded bytes from the socket and convert to string |
| <code>SendBytes(byte[] bytes)</code>            | Send a message   |
| <code>SendAsciiString(string message)</code>    | Send a string as ASCII encoded bytes                           |
| <code>SendUtf8String(string message)</code>     | Send a string as UTF-8 encoded bytes                           |
| <code>UdpClient()</code>                        | Constructor  |
| <code>UdpClient(int port)</code>                | Constructor. "port" is port number for receiving messages.     |

### 6.7.3 Example

#### 6.7.3.1 Sending Messages via UDP

UDP messages can be sent from RAPIX Logic. Here is an example of sending a UDP message when a Zone occupancy changes:

```
if (HasChanged(Zones["Relay 1"].IsOccupied))
{
    UdpClient MyUdpClient = new UdpClient();
    MyUdpClient.Connect("192.168.200.29", 55056);
    MyUdpClient.SendAsciiString(Zones["Test"].IsOccupied ? "Zone Occupied" : "Zone Vacant");
    MyUdpClient.Close();
}
```

### 6.7.3.2 *Receiving messages via UDP*

This example opens a UDP connection to another device then waits for messages containing "on" or "off" and sets the zone state accordingly.

In the global code:

```
UdpClient myUdpClient;
```

In the logic module (which runs every 0.5 seconds):

```
if (myUdpClient == null)
{
    myUdpClient = new UdpClient();
    myUdpClient.Connect("192.168.200.29", 55056);
    myUdpClient.SendAsciiString("Hello");
}

if (!myUdpClient.Connected)
{
    return;
}

string data = myUdpClient.ReadAsciiString();
if (string.IsNullOrEmpty(data))
{
    return;
}

if (data.ToLower().StartsWith("on"))
{
    Zones["Test"].On();
}
else if (data.ToLower().StartsWith("off"))
{
    Zones["Test"].Off();
}
```

## 6.8 TCP/IP Client

### 6.8.1 Properties

The TcpClient class properties are:

| Property Name | Type | Settable | Description                      |
|---------------|------|----------|----------------------------------|
| IsConnected   | bool | No       | Whether the client has connected |

### 6.8.2 Methods

The TcpClient methods are:

| Method                             | Description  |
|------------------------------------|--|
| Close()                            | Close the connection   |
| Connect(string hostName, int port) | Connect to the host at the specified port                      |
| byte[] ReadBytes()                 | Read bytes from the socket                                     |
| string ReadAsciiString()           | Read ASCII encoded bytes from the socket and convert to string |
| string ReadUtf8String()            | Read UTF-8 encoded bytes from the socket and convert to string |
| SendBytes(byte[] bytes)            | Send a message   |
| SendAsciiString(string message)    | Send a string as ASCII encoded bytes                           |
| SendUtf8String(string message)     | Send a string as UTF-8 encoded bytes                           |
| TcpClient()                        | Constructor  |

### 6.8.3 Example

#### 6.8.3.1 Sending Messages via TCP/IP

TCP/IP messages can be sent from RAPIX Logic. Here is an example of sending a TCP message when a Zone occupancy changes:

```
if (HasChanged(Zones["Room 1"].IsOccupied))
{
    TcpClient MyTcpClient = new TcpClient();
    if (MyTcpClient.Connect("192.168.200.29", 55056))
    {
        MyTcpClient.SendAsciiString(Zones["Test"].IsOccupied ?
                                    "Zone Occupied" :
                                    "Zone Vacant");
    }
    else
    {
        LogMessage("Connect to 192.168.200.29 failed");
    }

    MyTcpClient.Close();
}
```

## 6.9 Serial Port

For complete details of using the serial port, refer to "Zone Controller Serial Port Interface" Application Note available on the Ozuno web site.

### 6.9.1 Properties

The SerialPort class properties are:

| Property Name | Type | Settable | Description                     |
|---------------|------|----------|---------------------------------|
| IsOpen        | bool | No       | Whether the serial port is open |

### 6.9.2 Methods

The SerialPort methods are:

| Method   | Description  |
|--|--|
| void Close()   | Close the serial port  |
| bool Open()  | Open the serial port   |
| byte[] ReadBytes()   | Read bytes from the serial port  |
| byte[] ReadBytesUpToTerminator(char terminatorChar)  | Read bytes from the serial port up to the terminator character   |
| byte[] ReadBytesUpToTerminator(char terminatorChar1, char terminatorChar2)                       | Read bytes from the serial port up to the terminator characters  |
| byte[] ReadBytesUpToTerminator(char terminatorChar1, char terminatorChar2, char terminatorChar3) | Read bytes from the serial port up to the terminator characters  |
| string ReadAsciiLine()   | Read an ASCII string that is terminated by a carriage return and/or end of line (not including the terminators). |
| string ReadAsciiString()   | Read an ASCII string   |
| string ReadUtf8Line()  | Read a UTF string that is terminated by a carriage return and/or end of line (not including the terminators).    |
| string ReadUtf8String()  | Read a UTF string  |
| bool SendAsciiString(string message)   | Send an ASCII string   |
| bool SendBytes(byte[] bytes)   | Send bytes   |
| bool SendUtf8String(string message)  | Send a UTF-8 encoded string  |
| SerialPort(int comPort, int baudRate, int parity, int stopBits, bool xonXoff)                    | Serial port constructor (see below)  |

### 6.9.3 Constants

The SerialPort class constants are:

| Constant Name | Type | Value | Description                       |
|---------------|------|-------|-----------------------------------|
| PARITY_NONE   | int  | 0     | The serial port uses no parity    |
| PARITY_ODD    | int  | 1     | The serial port uses odd parity   |
| PARITY_EVEN   | int  | 2     | The serial port uses even parity  |
| PARITY_MARK   | int  | 3     | The serial port uses mark parity  |
| PARITY_SPACE  | int  | 4     | The serial port uses space parity |
| STOP_BITS_ONE | int  | 0     | The serial port uses one stop bit |

|                          |     |   |                                    |
|--------------------------|-----|---|------------------------------------|
| STOP_BITS_ONE_POINT_FIVE | int | 1 | The serial port uses 1.5 stop bits |
| STOP_BITS_TWO            | int | 2 | The serial port uses two stop bits |

#### 6.9.4 Constructor

The serial port constructor is:

```
SerialPort(int comPort, int baudRate, int parity, int
stopBits, bool xonXoff)
```

The Com port value is always 1. This is the serial port on the bottom of the RAPIX Zone Controller.

The baud rate, parity, stop bits and XON/XOFF settings need to match the device that is being communicated with.

#### 6.9.5 Example

##### 6.9.5.1 *Sending and Receiving Messages via Serial*

Serial port messages can be sent from RAPIX Logic. The example below sends a serial message when a Zone occupancy changes. It will also accept commands "on" and "off" from the serial port and control the Zone state.

Global code:

```
SerialPort MySerialPort;
```

In the logic module (which runs every 0.5 seconds):

```
// Make sure that the serial port exists and is open.
if (MySerialPort == null)
{
    MySerialPort = new SerialPort(1, 9600,
SerialPort.PARITY_NONE, SerialPort.STOP_BITS_ONE, false);
    MySerialPort.Open();
}

// If the Zone state has changed, send an update.
if (HasChanged(Zones["All"].State))
{
    if (Zones["All"].State)
    {
        MySerialPort.SendAsciiString("Zone is on\r\n");
    }
    else
    {
        MySerialPort.SendAsciiString("Zone is off\r\n");
    }
}

// See if a command has been received.
string input = MySerialPort.ReadAsciiLine();
```

```
if (input.Length == 0)
{
    return;
}

if (input == "on")
{
    Zones["All"].On();
}
else if (input == "off")
{
    Zones["All"].Off();
}
else
{
    MySerialPort.SendAsciiString("Unknown command: " + input +
"\r\n");
}
```

## 6.10 Other Properties and Methods

### 6.10.1 Date and Time

Date and Time properties include:

| Property Name        | Type     | Settable | Description                                  |
|----------------------|----------|----------|--|
| Now                  | DateTime | No       | The current date/time                        |
| IsDaylightSavingTime | bool     | No       | Whether daylight saving is currently active  |
| Sunrise              | DateTime | No       | The sunrise time today                       |
| Sunset               | DateTime | No       | The sunset time today                        |
| StartupTime          | DateTime | No       | The Date/Time that the Logic started running |
| RunTime              | TimeSpan | No       | How long the logic has been running          |

### 6.10.2 Connectivity

Connectivity properties include:

| Property Name        | Type | Settable | Description   |
|----------------------|------|----------|---|
| EthernetConnected    | bool | No       | Whether the Zone Controller Ethernet is connected           |
| RapixClientConnected | bool | No       | Whether a RAPIX client is connected to the Zone Controller  |
| ModbusConnected      | bool | No       | Whether a Modbus server is connected to the Zone Controller |

### 6.10.3 Logic Module

Logic module methods include:

| Method                            | Description                   |
|-----------------------------------|-------------------------------|
| void DisableModule(string name)   | Disable the logic module      |
| void EnableModule(string name)    | Enable the logic module       |
| bool IsModuleEnabled(string name) | True if the module is enabled |

Logic Module properties include:

| Property Name | Type | Settable | Description  |
|---------------|------|----------|--|
| IsFirstRun    | bool | No       | Whether this is the first time that the logic module has run |

### 6.10.4 Level Conversion

| Method  | Description  |
|---|--|
| int LevelConvert.DaliLevelToIntPercent(int level)   | Convert from DALI Level (0 – 254) to percent (0 – 100) |
| double LevelConvert.DaliLevelToPercent(int level)   | Convert from DALI Level (0 – 254) to percent (0 – 100) |
| int LevelConvert.PercentToDaliLevel(double percent) | Convert from percent (0 – 100) to DALI Level (0 – 254) |

### 6.10.5 Other

Other methods include:

| Method                                    | Description   |
|---|---|
| <code>void LogMessage(string text)</code> | Write an Information level message to the Zone Controller log |

If the logged message is longer than 250 characters, it will be truncated. Any double quote characters will be changed to single quote characters.

### 6.11 Constants

The following constants are defined:

```
const bool OFF = false;
const bool ON = true;
const bool NO = false;
const bool YES = true;
const bool CLEAR = false;
const bool SET = true;
```

This allows for clarity in the code. For example, the following are all equivalent:

- `if (FlagGroups["Sensors"].Flags["Office"].State)`
- `if (FlagGroups["Sensors"].Flags["Office"].State == ON)`
- `if (FlagGroups["Sensors"].Flags["Office"].State == YES)`
- `if (FlagGroups["Sensors"].Flags["Office"].State == SET)`
- `if (FlagGroups["Sensors"].Flags["Office"].State != CLEAR)`
- `if (FlagGroups["Sensors"].Flags["Office"].IsSet)`

## 7 Best-Practices

### 7.1 Comments and naming

Put lots of comments in the code. They are invaluable if changes are required at a later date.

Use clear variable names. For example, a variable called "counter" is much better than one called "x".

### 7.2 Module Period

Use the longest module period suitable for the application. If something needs to be performed every few minutes, there is no need to have a period of ½ second – it just adds load to the processor.

### 7.3 Performing Actions on Changes

It is important to try to write Logic code that performs actions only when something changes.

For example, if you want the level of Zone 2 to track the level of Zone 1, the temptation is to have a module run every 250ms that does this:

```
Zones[2].Level = Zones[1].Level;
```

The problem with this is that it may result in commands being sent to DALI 4 times every second setting the level of Zone 2. This could be very disruptive of the correct operation of the system.

The Zone Controller does anticipate this situation by only setting a Zone level if the requested level is not the same as the current level. However, this is not fool-proof. For example, if you have a Zone that only contains relays, then its level will be either 0% or 100%. If your Logic code sets the Zone level to 50%, then the DALI commands will always be sent out to the DALI Lines.

The best way to ensure that actions are performed only when something changes is to test for the change before performing the actions. For the example above, the code would be:

```
if (HasChanged(Zones[1].Level))
{
    Zones[2].Level = Zones[1].Level;
}
```

Alternatively, the "early return" coding style can be used:

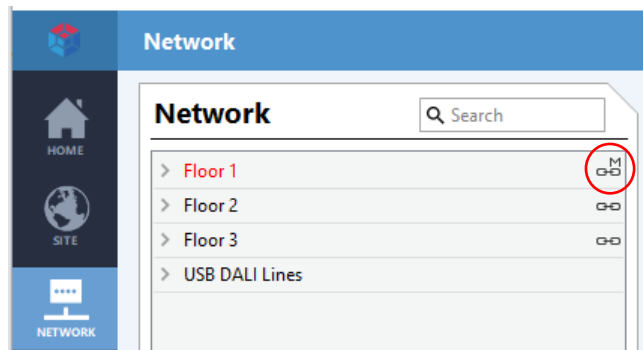
```
if (!HasChanged(Zones[1].Level))
{
    return;
}
Zones[2].Level = Zones[1].Level;
```

You can see that the code exits if the level of Zone 1 has not changed. The following code will only be run when the level of Zone 1 changes.

## 8 Debugging

Logic runs in the Master Zone Controller. The Master is usually the one with the lowest IP Address.

The Master can be seen on RAPIX Integrator with the M next to the link icon.



***"Floor 1" is the Master***

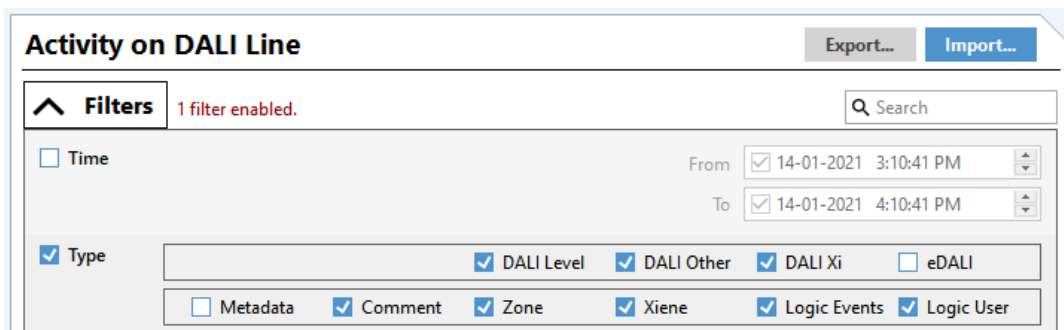
Alternatively, the Zone Controller menu item 1.12.10 shows whether a Zone Controller is Master or Slave.

If the logic does not appear to be running:

1. Find which Zone Controller is the Master.
2. Navigate to the menu item 1.12.12
3. The display should show "Running". If not, there will be a message with the reason why the logic is not running. Logic will only run:
  - a. If the date and time has been set
  - b. When the DALI Line scan is complete

Using logging:

1. Use the logic LogMessage() method to log when the logic performs various actions.
2. Transfer the project to the Zone Controllers.
3. Open the RAPIX Integrator log form.
4. Select the filter option to show **Logic Events** and/or **Logic User** messages (the ones written using the LogMessage method).



5. Start the log running (make sure the Master Zone Controller is selected)
6. You will see logic messages in the log.

## 9 Examples

### 9.1 Zone Toggle

To toggle the on/off state of a Zone called "Room 2":

```
Zones["Room 2"].State = ! Zones["Room 2"].State;
```

### 9.2 Zone Level Adjustment

To increase the level of a Zone called "Room 2" by 10%:

```
Zones["Room 2"].Level = Zones["Room 2"].Level + 10%;
```

To decrease the level of a Zone called "Room 2" by 10%:

```
Zones["Room 2"].Level = Zones["Room 2"].Level - 10%;
```

To decrease the level of a Zone called "Room 2" by 10%, but prevent it from turning off entirely:

```
if (Zones["Room 2"].IsOn)
{
    int new_level = Zones["Room 2"].Level - 10%;

    // If the new level is > 0, then use the new level.
    // Otherwise, use level 1.
    if (new_level > 0)
    {
        Zones["Room 2"].Level = new_level;
    }
    else
    {
        Zones["Room 2"].Level = 1;
    }
}
```

A more concise way to do the same thing is to use the C# ternary conditional operator:

```
if (Zones["Room 2"].IsOn)
{
    int new_level = Zones["Room 2"].Level - 10%;

    // If the new level is > 0, then use the new level.
    // Otherwise, use level 1.
    Zones["Room 2"].Level = new_level > 0 ? new_level : 1;
}
```

### 9.3 Zone Tracking

To get the state of one zone ("Room 2") to track the state of another zone ("Room 1"):

```
if (Zones["Room 2"].State != Zones["Room 1"].State)
{
    Zones["Room 2"].State = Zones["Room 1"].State;
}
```

### 9.4 Conditional Zone Tracking

To get the level of one zone to track the other if a flag is set:

```
if (FlagGroups["Room Divider"].Flags["Room 1 and 2"].State &&
    Zones["Room 2"].Level != Zones["Room 1"].Level)
{
    Zones["Room 2"].Level = Zones["Room 1"].Level;
}
```

### 9.5 Logical OR

To have Zone 3 on if Zone 1 OR Zone 2 is on:

```
Zones[3].State = Zones[1].State || Zones[2].State;
```

### 9.6 Logical AND

To have Zone 3 on if Zone 1 AND Zone 2 is on:

```
Zones[3].State = Zones[1].State && Zones[2].State;
```

### 9.7 Logical NOT

To have Zone 2 the opposite state of Zone 1:

```
Zones[1].State = !Zones[1].State;
```

## 9.8 Adjacent Area Control

To keep a corridor on when an adjacent office is occupied:

Global code:

```
int OccupancyCounter = 0;
```

Module:

```
// Check whether the corridor needs to be turned on.
if (Zones["Office"].IsOccupied && Zones["Corridor"].IsAnyOff)
{
    Zones["Corridor"].On();
    OccupancyCounter = 300;
}

// Check whether the corridor needs to be turned off.
if (!Zones["Office"].IsOccupied && Zones["Corridor"].IsAnyOn
&& OccupancyCounter > 0)
{
    OccupancyCounter--;
    if (OccupancyCounter == 0)
    {
        Zones["Corridor"].Off();
    }
}
```

## 9.9 Initialisation

There are several ways of initialising aspects of the logic.

### 9.9.1 Global Declaration

Variables can be initialised in the global code. For example:

```
int MyCounter = 0;
RapixTimer EventTimer = new RapixTimer();
```

### 9.9.2 Run Once Module

A module can be used to do initialisation. The best way to do this is:

1. Name it clearly (e.g. "Initialisation" or "Startup")
2. Make sure it runs before all other modules (set its Delay after logic starts to 0, and all other modules to 1 second)
3. Set it to run once only

The screenshot shows two panels from a software interface. The top panel, titled 'Properties', contains the following settings: 'Name' is 'Startup'; 'Include' is checked and 'Catch-up on Power-up' is unchecked; 'Run once' is selected from a dropdown menu; and 'Delay after logic starts' is '0' from a dropdown menu. The bottom panel, titled 'Script', contains the following code:

```
1 // Initialise everything.
2 myTcpClient = new TcpClient();
3
```

### 9.9.3 Using IsFirstRun

The IsFirstRun property will be true only the first time that the logic runs. That allows you to put initialisation code at the start of the module. For example:

```
if (IsFirstRun)
{
    // Initialise everything.
    myTcpClient = new TcpClient();
}
```

## 9.10 Using Date and Time

### 9.10.1 Time of Day

To perform an action at 7:30PM, the code will be:

```
if (NowTrue(Now.TimeOfDay >= new TimeSpan(19, 30, 0)))
{
    // Perform action
}
```

Do not write the code like this:

```
if (Now.TimeOfDay == new TimeSpan(19, 30, 0))
{
    // Action might get missed, or executed multiple times!!!
}
```

In this case, the action may get missed if the logic module does not get executed at exactly 7:30 PM. It could also get run more than once if the logic module is executed multiple times within the second at 7:30:00 PM

The NowTrue method in the example code ensures that the action is performed exactly once when the time changes from before 7:30PM to 7:30PM or after.

### 9.10.2 Day of Week

To execute some logic only on Saturdays:

```
if (Now.DayOfWeek == DayOfWeek.Saturday)
{
    // Put logic code here
}
```

### 9.10.3 Day of Year

To execute an action on 25<sup>th</sup> December every year at 6AM:

```
if (NowTrue(Now.Day == 25 && Now.Month == 12 && Now.Hour == 6))
{
    // Perform action
}
```

Note that the NowTrue function is used in this instance because the Now.Hour will equal 6 for a whole hour and the module could be executed thousands of times during that interval, but we only want the action to be performed once.

### 9.10.4 Sunrise/sunset

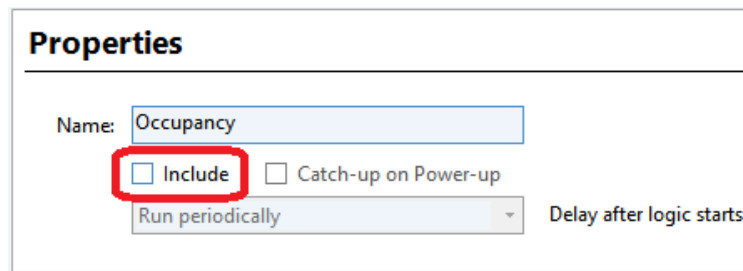
To perform an action an hour after sunrise:

```
if (NowTrue(Now > Sunrise.AddHours(1)))
{
    // Perform action
}
```

## 9.11 Enabling / Disabling Logic Modules

### 9.11.1 Excluded Modules

Individual Logic modules can be permanently disabled by de-selecting the "included" option:



**Properties**

Name:

☒ Include ☐ Catch-up on Power-up

Delay after logic starts:

### 9.11.2 Enable / Disable methods

The EnableModule and DisableModule methods can be used to dynamically enable and disable modules. A module is also able to disable itself using DisableModule.

For example, a module could be used to enable another module based on the state of an Xi Operating property. In the example below, the "Occupancy" module is to be enabled only during non-peak hours:

```
if (OpProps["Building Mode"].IsValue("Non-peak hours"))
{
    DisableModule("Occupancy");
}
else
{
    EnableModule("Occupancy");
}
```

### 9.11.3 Conditional Operation

The simplest method of disabling a module under certain conditions is to just include the condition as part of the module code.

For example, we may want to perform an action only if an Xi Flag is set:

```
if (FlagGroups["Sensor enable"].Flags["Floor 1"].IsSet)
{
    // Perform action
}
```

An alternative way of writing this is the "early return" coding style where the opposite condition (flag is clear in this case) is used to exit the module at the start:

```
if (FlagGroups["Sensor enable"].Flags["Floor 1"].IsClear)
{
    return;
}

// Perform action
```

There are some advantages to using the early return style. It separates the condition from the rest of the module code.

It also allows the use of delays in the rest of the module code. For example, this is valid:

```
if (FlagGroups["Sensor enable"].Flags["Floor 1"].IsClear)
{
    return;
}

// Perform action 1
Delay(2);
// Perform action 2
```

**This will not work** because the delay is inside the code block:

```
if (FlagGroups["Sensor enable"].Flags["Floor 1"].IsSet)
{
    // Perform action 1
    Delay(2);
    // Perform action 2
}
```

## 9.12 Sequence of Events

This example uses a Zone being turned on to trigger a sequence of scenes with delays between them:

```
if (!NowTrue(Zones[1].IsOn))
{
    // No trigger yet, so exit.
    return;
}

Scenes[1].Set();
Delay(2);
Scenes[2].Set();
Delay(2);
Scenes[3].Set();
```

Note that **you cannot write the code like this** because delays are not allowed inside a block:

```
if (NowTrue(Zones[1].IsOn))
{
    // *** This will not work ***
    Scenes[1].Set();
    Delay(2);
    Scenes[2].Set();
    Delay(2);
    Scenes[3].Set();
}
```

Another way to achieve this is to have one module used for running the sequence and have it start disabled (select the "Don't run automatically" option). Then use another module to enable the sequence.

Module 1 ("Trigger"):

```
if (NowTrue(Zones[1].IsOn))
{
    EnableModule("Sequence");
}
```

Module 2 ("Sequence") – initially disabled:

```
Scenes[1].Set();
Delay(2);
Scenes[2].Set();
Delay(2);
Scenes[3].Set();
```

If there is a long sequence of actions, it is tempting to write a "for" loop to iterate over all of them. However, delays cannot be used inside loops. To run a sequence of 20 scenes, it could be done like this:

Global code:

```
int SceneNumber = 0;
```

Module 1 ("Trigger"):

```
if (NowTrue(Zones[1].IsOn))
{
    EnableModule("Sequence");
    SceneNumber = 0;
}
```

Module 2 ("Sequence") – initially disabled:

```
SceneNumber++;
if (SceneNumber > 20)
{
    // Finished
    DisableModule("Sequence");
    return;
}

Scenes[SceneNumber].Set();
Delay(2);
```

If the scenes do not have consecutive id numbers, then they can be indexed by name. The example above is the same, except with a small change to the "Sequence" module:

```
SceneNumber++;
if (SceneNumber > 20)
{
    // Finished
    DisableModule("Sequence");
    return;
}

string SceneName = "Scene Sequence " + SceneNumber;
Scenes[SceneName].Set();
Delay(2);
```

### 9.13 Handling Error Conditions

If the Ethernet connection to the Zone Controller stops working, it will not be possible for a building management system (BMS) to control the RAPIX system through the Zone Controller. It may be desirable to turn all lights on in this situation. The logic code to do this could be:

```
// If the Ethernet is disconnected for 10 seconds, then turn
on all DALI Lines.
if (StayedFalse(EthernetConnected, 10))
{
    Zones["All Lines"].On();
}
```

If there is a failure with the BMS itself, then the above code will not work, since the Ethernet would remain connected. A more robust solution would be for the BMS to send a "heart-beat" of some sort and have the logic detect if it has stopped.

One way to do this is to have the BMS toggle the state of a virtual zone on a regular basis, say every minute. If the logic observes the virtual zone staying off or on for more than a minute, then it can turn on the lights. One simple way to solve this is using code similar to the example above. Another is to use timers:

In global code:

```
RapixTimer MyTimer = new RapixTimer();
```

In module:

```
// Detect if the "heart-beat" from the BMS has stopped.

// If the timer is not running, then start it.
if (!MyTimer.Running)
{
    MyTimer.Start();
}

// If the Zone changes state, then re-start the timer.
if (HasChanged(Zones["Virtual Zone"].State))
{
    MyTimer.Start();
}

// If the timer gets to 90 seconds then the "heart-beat" has
stopped and the lights need to be turned on.
// We only want to do this once, when the timer first gets to
90 seconds.
if (NowTrue(MyTimer.Time > 90))
{
    LogMessage("BMS heart-beat has failed. Turning lights on.");
    Zones["All Lights"].On();
}
```