ESR Consortium
HomeEZ-1.0

The HomeEZ language
Profile Specification

ESR023

Reference: ESR-SPE-023-HomeEZ
Version: 1.0
Rev: A

Trademarks

Contents

Tables

Illustrations

1 PREFACE TO *HOMEEZ* SPECIFICATION

This document defines the *HomeEZ language specification* v 1.0.

1.1 Who should use this specification

This specification is targeted at the following audiences:

- Implementors of the *HomeEZ* language,
- Smart home services developers that wish to design smart offering for customers that are part of the rich *HomeEZ* ecosystem,
- Smart home devices engineers that wish to provide the *HomeEZ* rich ecosystem with new sensors/activators capabilities.

1.2 How this language specification is organized

This specification is organized as follows:

- Introduction: Explains what *HomeEZ* language is and why it has been designed. It presents the main advantages and general perspectives of *HomeEZ* .
- Basic Concepts: Aims at making the reader familiar with the fundamental *HomeEZ* notions and vocabulary.
- Exact semantic of all concepts : the syntax of the language, the manipulated first class concepts, the engine, the life-cycle of an application.

1.3 Implementation notes

This *HomeEZ* specification does not include any implementation considerations. *HomeEZ* implementors are free to use whatever techniques they deem appropriate to implement the specification. *HomeEZ* experts have taken great care not to mention any special features, in order to encourage fair competing implementations.

2 INTRODUCTION

The HomeEZ language allows one to write scenario for smart markets, such as Smart Home, so its name. It is designed to be simple so that no particular notion about software programming is needed to achieve fluency in the language. Its purpose is to handle and to describe automation applications. It has two levels of descriptive power: the first one deals with straight forward objects (naming, linking, binding actions on events), while the second one provides a more expert semantic with classes that organize manipulated concepts.

The HomeEZ programming language is not typed and has an XML based syntax. The concepts provided by HomeEZ are :

- automatic pool of devices management
- construction and manipulation of direct acyclic graphs of devices (name space)
- events triggering (time, asynchronous (hardware) events, ...) and action invocation
- atomic states of devices management and variables manipulation
- scenario instances and scenario classes
- expressions computation model

Next a simple example of HomeEZ.

```
<homeez>
        <scenario>
                <body>
                        <invoke device="Light" action="toggle" />
                </body>
                <trigger>
                        <event device="Button" name="pressed" />
                </trigger>
        </scenario>
</homeez>
```

The above first example defines an anonymous scenario. It declares: “*Each time the "Button" is pressed, then toggle the "Light"* ”.

3 HOMEEZ BASIC CONCEPTS

3.1 Name space

HomeEZ defines three global name spaces, one for each kind of major element type. A name space is defined for:

- devices and groups
- scenario classes
- scenario instances

Two elements may not have identical names in the same name space. When an element has no name, it is said to be an anonymous element. Names can be defined using simple identifiers, or using fully qualified names, allowing to sort elements into categories. Fully qualified names are separated by dots '.' (for example 'test.hez.scenario'). It is generally advised to use fully qualified names to define elements, to avoid conflicts with other scenarios or devices, as the name spaces are global.

3.2 Devices

3.2.1 Definition

A device logically represents within the HomeEZ system an entity that may interact with the rest of the system using 3 different mechanisms:

- Triggering events: when an event is issued by a device, the event holds the device as its source.
- Performing actions: although the exact meaning of an action is device specific, it is assumed that an action has a side-effect on the device itself therefore making physical actions on the real world through the physical device capabilities
- Holding a state: the sum of status fields. Reading a device state must have no side-effect on the device. It is possible to read more than one device status at a time in the HomeEZ language, to ensure the consistency between read values. A state can be written by the device itself according to its own internal logic, or also within the HomeEZ language when fields are writable. Field types may be base types or strings.

3.2.2 Device manager

The aim of HomeEZ is to allow the management of devices from various vendors. The engine therefore exposes an API that allows vendors to register themselves as device managers. A device manager holds a list of devices, and can be queried for all available devices, or for a precise device. The only constraint for the vendors is to provide unique identifiers (UIDs) for their devices, that won't vary in time. This UID will then be used for device pairing.

3.2.3 Device pairing

Pairing a device to the HomeEZ system requires 2 mandatory attributes:

- A unique identifier (UID): this identifier is used to identify the actual device from a provider. When pairing a new device, all device managers will be queried for a device with the given UID
- A name: this identifier can then be used throughout scenarios to refer to the device. A device must not have the same name as another device or group.

The whole pairing system only relies on associating a user's readable name to a device manager UID.

3.2.4 Device groups

Devices may be sorted inside groups. A group is defined by a unique name, which can be used in scenarios in place of a device name. It is also possible to sort a group inside another group, by defining a “parent” group. Groups may then be considered as virtual devices, as they implement all the mechanisms of a device:

- Triggering events: events triggered by devices contained in a group are also triggered by the group and all its parents. It is then possible to trigger a scenario execution on an event triggered by a group of devices.
- Performing actions: executing an action on a group of devices will trigger the action on every device of the group able to execute it.

- Holding a state: when querying a group for a state, all the devices of the group will be queried for the state. The result is then computed using the “merge” attribute, allowing to provide various computation methods (for example min, max, mean, …) to determine the corresponding status of the group.

Devices or groups may be included in more than one parent group, at the condition that the dependency graph is acyclic.

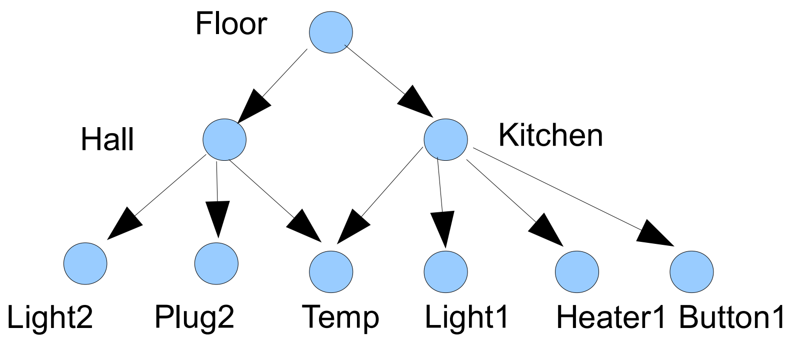

Illustration 3-1: 9 devices, with 3 groups that materializes locations. The “Temp” sensor is shared along the two rooms “Kitchen” and “Hall”.

Performing an action (or querying a state) on a group is handled by first flattening the list of all edible devices (meaning that each device is only present once in the list, even if included in many sub-groups), and the action is then performed on each device. Each action will then be triggered only once on each device, and each device will have the same weight for the computation of the final merged status.

By default a “Home” group is defined, including all devices.

3.2.5 Device state

Every device can be set into an active mode, or into a passive mode. In the passive mode, a device does neither trigger events, nor perform actions. Its state remains available though.

3.2.6 Devices serialization

If only a name and an UID are necessary to pair a device, a more complete syntax is provided to allow the description of a device. The engine is thus able to provide a feedback on the available devices, or to persist its internal configuration to a human readable file. This syntax includes available actions, statuses and events. When querying the engine for devices, already paired devices may be identified by their name being set. This syntax may also be used to define virtual devices.

3.3 Scenarios & statements

3.3.1 Description

A scenario represents a series of actions done by the home devices. It makes a relationship between a fact and one or several actions. It may require no, one, or more parameters for being launched. Scenarios are split into 3 main concepts:

- Standalone scenario: a scenario implementing a simple list of actions, with no parameters, that directly refer to devices or groups
- Scenario classes: a parameterizable and reusable scenario, defining parameters for all its external dependencies and that may then be instantiated.
- Scenario instances: an instance of a scenario class, providing its required parameters.

Scenario classes are defined using the <scenarioClass> mark, whereas standalone scenarios and scenario instances are both defined using the <scenario> mark.

3.3.2 States

A scenario has different states: active, inactive and queued (illustrated on the picture below). When a scenario is added to the system, it goes by default from the initialization step to the active mode. The scenario becomes active, but it can be unlinked to go back on the initialization step. An active scenario can be suspended. A scenario suspended is on inactive mode and can not be triggered.

An inactive scenario can be resumed to become active again. Active scenario can also be scheduled for going on the queued state. A queued scenario can be killed and go back on active mode.

If a queued scenario execution fails, it is killed and unlinked.

Below is the complete execution scenario possible states:

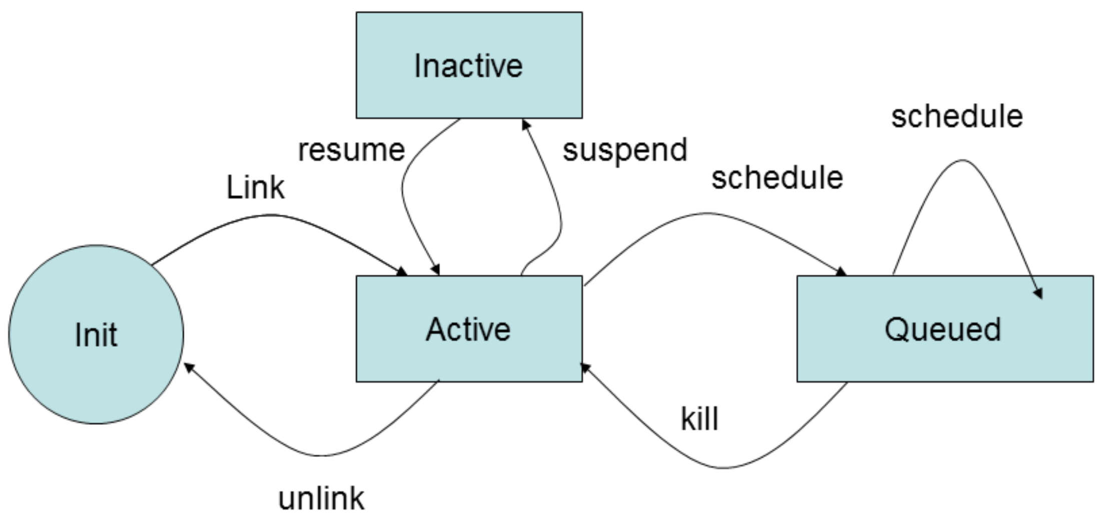

Illustration 3-2: Execution scenario state

3.3.3 Scheduling

Only one scenario executes itself at a time. When a scenario starts executing itself, it is guaranteed that the device and scenario topologies will not change (the changes are postponed to the end of the scenario's execution). A scenario is never blocked, it can only run to it's ending (and possibly re-schedule itself). A scenario is always triggered by some event. This event can be queried to get the source that triggered the event, and also to get the time-stamp of the event. The picture below shows scenarios in different phases in time:

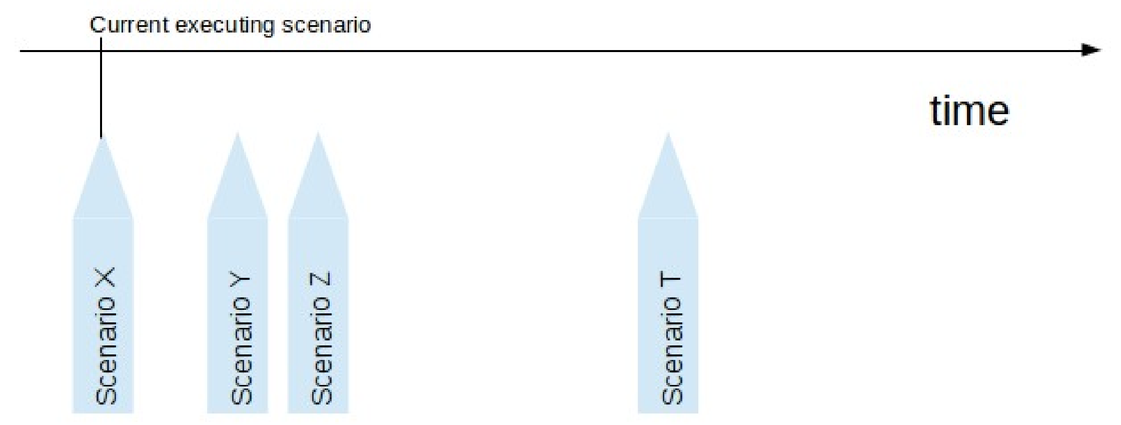

Illustration 3-3:Time execution phase

Statements used by a scenario to manage scenario scheduling are: **trigger, event, timer** and **monitor.** These will be presented on the general syntax section.

3.3.4 Variable scopes

The variables are scoped depending on the kind of scenario:

- Standalone scenario: a standalone scenario may access the global scope of devices and groups, and defines a local scope containing local computation results
- Scenario class: a scenario class may not access the global scope, but defines a local scope containing its parameters and the local computation results. A scenario class does not depend of any external context.
- Scenario instance: a scenario instance may access the global scope to provide argument to the instantiated class, but does not define any local scope.

As for global scopes, two variables may not have the same name in a local scope. If a local scope variable has the same name than a global scope variable, the local scope variable takes the precedence and shadows the global variable. It is therefore possible to shadow global variables by creating local variables with the same name.

3.3.5 Action invocation

Each device described in the HomeEZ language can perform actions according to its definition. This call to a specific implemented method is an action invocation. It is possible using the <invoke> tag defined in the HomeEZ language. According to the method definition, this tag can have several arguments. It will be presented on the general syntax section.

3.3.6 Scenario print

A scenario can be asked for its pretty-print. The pretty print is the readable description in a human language, here English, of the use case of the scenario.

3.4 Expression

HomeEZ allows using a "usual programmatic syntax" for basic types and string. The following operators are allowed:

Unary operators (without string):

! ~ + -

Binary operators (without string):

+ - / % * >> >>> << & | ^ && || > < >= <= == !=

Operators for string:

The special character # allows to call string operators, to apply some changes on string:

For example: #upper("nantes") returns "NANTES".

Status access operator:

It is possible to read the status of a device with the status access operator '.'.

For example: "device.status" returns the value of "status" in the "device".

3.5 Event driven and time

Events are broad casted to the whole system. Each event has:

- a name: the name of the event,
- a source: the device or the scenario that generated the event

An event is "time-stamped" with a "time" that permits to order events. Two events have for sure a different time-stamp.

In this way, several other notions are defined in order to organize the scenario triggering events such as:

- timer: it defines the scenario time intervals,
- cron: it defines a periodic scenario execution,
- basic event: it is triggered asynchronously.

4 THE HOMEEZ LANGUAGE

The HomeEZ language uses both lexical and syntax from XML for all its description, except for the manipulation of arithmetical expressions where “a usual programmatic syntax” is used.

It is a not typed programming language. In this model, the actions' names of objects are specialized. All objects are recognized by the operations they are able to do. The main goal of the home automation model is to declare what I want to do (which actions) without necessarily knowing what can do it (which device). In this way, the device change becomes transparent for the language that is a high level description.

4.1 Compilation Unit

4.1.1 Semantic

Each element of HomeEZ language is defined by a syntactical goal <homeez>. Which compilation units are accessible at compile-time is determined by the host system. Each compilation unit is compiled into a standalone binary file.

4.1.2 Syntax

- <homeez>: the root element of one HomeEZ element description
    - attributes:
        - name (optional): defines the content of the HomeEZ element description.
        - version (optional)**:** provides the version as x.y.z respectively major, minor and patch.
    - Child elements: <configuration>, <scenarioClass>, <scenario>.

4.2 Configuration

4.2.1 Semantic

The configuration is the HomeEZ representation of all devices configured in the system.

4.2.2 Syntax

- <configuration>: element that defines the list of devices configured in the system.
    - Attributes: none.
    - Child elements: <device>, <group>.

4.3 Device

4.3.1 Semantic

A device represents a component that can make an action, hold a status and raise events. This syntax is used for both pairing a device to the system, or persisting devices present on the system.

4.3.2 Syntax

- <device>: element that represents a device usable on a specific location.

- attributes:
  - name: defines the name of the device.
  - uid: represents the unique identifier of a device.
  - includedIn (optional): allows setting the group of the device.
- Child elements: <status>, <action>, <event>, <group>.

4.3.3 Status element

- <status>: element that represents the status of a device
  - attributes:
    - name: defines the name of the status
    - value: represents the value of the status
  - Child elements: none.

4.3.4 Action element

4.3.4.1 Syntax

- <action>: element that represents an action associated to a device that can be invoked by a scenario
  - attributes:
    - name: defines the name of the action
  - Child elements: <param>

4.3.4.2 Param element

- <param>: element that represents the parameter associated to an action.
  - attributes:
    - name: defines the name of the parameter
  - Child elements: none

4.3.5 Event element

- <event>: element that represents an event that a device may rise
  - attributes:
    - name: defines the name of the event
  - Child elements: none

4.4 Group

4.4.1 Semantic

A group defines a bunch of devices or groups that can be used in scenarios. Groups can be declared at configuration level (to declare a group and its parents), or at device level (to specify that the device belongs to the group). A group doesn't need explicit declaration. If a device is included in a non-existing group, or a group parent is set to a non-existing group, it is implicitly declared.

4.4.2 Syntax

- <group>: element that represents a group
  - attributes:
    - name: defines the name of the group
    - parent (optional): parent group
  - Child elements: none.

4.5 Scenario Class

4.5.1 Semantic

The scenario class is a series of implemented actions available and provided by the system. It will be possible to get new scenario classes from a scenario store for having more choices on the sequence of actions.

4.5.2 Syntax

- <scenarioClass>: element that represents a scenario class.
  - attributes:
    - name: defines the fully qualified name of the scenario class.
  - Child elements: <param>, <body>, <trigger>.

4.5.3 Param element

- <param>: element that represents a parameter used by the scenario class.
  - attributes:
    - name: defines the name of the parameter.
  - Child elements: none

4.5.4 Body element

- <body>: element that regroups the actions list of a scenario class.
  - Attributes: none
  - Child elements: <monitor>, <invoke>, <expression>, <if>

4.5.4.1 Monitor element

- <monitor>: element that allows to monitor a scenario.
  - attributes:
    - scenario: represents the name of the scenario to monitor. The reserved value “this” may be used to refer to the current scenario.
    - value: defines how to monitor. The allowed values are: schedule, kill, resume and suspend. Anonymous scenario (without name) cannot be monitored.
    - delay: represents a delay for scheduling a scenario.
  - Child elements: none

4.5.4.2 Schedule, kill, resume, suspend elements

Schedule, kill, resume and suspend elements are aliases for monitor elements (see §4.5.4.1), with "value" attribute respectively set to "schedule", "kill", "resume" and "suspend".

4.5.4.3 Invoke element

4.5.4.3.1 Syntax

- <invoke>: element that allows to call an action on a device.
  - attributes:
    - device: represents the name of the device or group.
    - action: represents the name of the invoked action.
    - result (optional): represents a variable name to store the result returned by the method invoked.
    - merge (optional): represents the operation to apply on a series of results from several devices of the same group. By default, the mean of the results is used. Default operations are available: min, max, mean and std that are respectively the minimum, the maximum, the mean and the standard deviation. The list of operations can be extended via an API in the engine.
  - Child elements: <arg>

4.5.4.3.2 Arg element

- <arg>: element that represents the mandatory arguments of the invoked action.
  - attributes:
    - name: represents the name of the argument.
    - value: represents the value of the argument.
  - Child elements: none

4.5.4.4 Status element

4.5.4.4.1 Syntax

- <status>: element that allows to read or write the status of a device.
  - attributes:
    - device: represents the name of the device or group.
    - name: represents the name of the read status.
    - value (optional): represents the value to write in the status field.
    - result (optional): represents a variable name to store the read status.
    - merge (optional): represents the operation to apply on a series of results from several devices of the same group. By default, the mean of the results is used. Default operations are available: min, max, mean and std that are respectively the minimum, the maximum, the mean and the standard deviation. The list of operations can be extended via an API in the engine.
  - Child elements: <status>: status accesses can be nested, insuring that the read/write operations will be performed synchronously, and that all the read/written values will be consistent with each other.

4.5.4.5 Expression element

- <expression>: element that represents an expression.
  - attributes:
    - value: represents the expression that will be evaluated
    - result: represents the variable name in which the expression evaluated will be stored.
  - Child elements: none

4.5.4.6 If, then, else elements

- <if>: element that evaluates a boolean condition. The series of actions to do when the condition is satisfied are between the tag <then>, and between the tag <else> when the condition is not satisfied.
  - attributes:
    - condition: represents a boolean expression.
  - Child elements: <then>, <else>

For child elements of tags <then> and <else>, see §4.5.4.

4.5.5 Trigger element

- <trigger>: element that represents a series of conditions that trigger a scenario. It is different from the tag <if> where the condition is an expression. Here, the condition is an event or a timer. The scenario is triggered when any of the conditions are met.
  - attributes: no attributes
  - Child elements: <timer>, <event>

4.5.5.1 Timer element

4.5.5.1.1 Syntax

- <timer>: element that represents a time interval applied to the scenario. It can be used for representing the periodic execution of a scenario.
  - Attributes:
    - startAt (optional): defines the periodic date the timer begins to be active. See §4.5.5.1.2 for format detail.
    - endAt (optional): defines the periodic date the timer stops being active. See §4.5.5.1.2 for format detail.
    - date (optional): defines the periodic date the timer is triggered. See §4.5.5.1.2 for format detail.
    - delay (optional): represents the periodic value defined in milliseconds. This field is not compatible with the “date” field.
  - Child elements: none

4.5.5.1.2 Date format

The date format for the “startAt”, “endAt” and “date” attributes is the following:

yyyy MM dd HH mm ss

With the following values:

- yyyy: the 4 digits year number
- MM: the month number (1 for January to 12 for December)
- dd: the day number (from 1 to 31). It is also possible to reference the number of the day in week with the prefix 'd' (from d1 for Monday to d7 for Sunday).
- HH: the hour of the day (from 0 to 23)
- mm: minute in hour (from 0 to 59)
- ss: second in minute (from 0 to 59)

The date syntax is flexible as it allows:

- Replacing the “space” separator by a “colon” (for example “2014:01:01 00:00:00” is a valid date)
- Omitting date elements from left to right (for example “01:01 00:00:00” represents the first January, at midnight, for every year)

Each element of the date also supports the following syntax:

- '*': all possible values (for example “01 *:00:00” represents every hour, the first day of the month)
- '-': a value range (for example “15-17:00:00” represents 15h, 16h and 17h)
- '/': value range step (for example “10-20/3:00:00” represents 10h, 13h, 16h and 19h).
- ',': list possible values (for example “15,17,20:00:00” represents 15h, 17h and 20h)

4.5.5.2 Event element

- <event>: element that represents an event issued by a device.
  - Attributes:
    - device: represents the name of the device or group triggering the event
    - name: represents the name of the triggered event.
  - Child elements: none

4.6 Scenario

4.6.1 Semantic

A scenario is either an instance of a scenario class, or simply a series of action directly configured by a user (called a “standalone scenario”). A scenario can be anonymous (without name) and it is possible to trigger it by another scenario.

4.6.2 Syntax

- <scenario>: element that represents an instance of a scenario class or a standalone scenario.
  - attributes:
    - name (optional): defines the name of the scenario. It is empty for anonymous scenario.
    - class (optional): represents the name of the scenario class to which it refers.

    - initialState (optional): initial state of the scenario when added to the system. May be either “init”, “active”, “inactive” or “queued”. Default is “active”.
  - Child elements: <arg>, <body>,<trigger>.

Scenario instances must have arguments and no body, whereas standalone scenarios must have a body and no arguments. For child elements descriptions, see §4.5.4.3.2 (arg), §4.5.4 (body) and §4.5.5 (trigger).

4.7 Expressions syntax

4.7.1 Expression types

HomeEZ supports the following expression types:

- Integers (arithmetic operators)
- Strings (string operators)

4.7.2 Variable access

Any local or global variable may be read within expressions, according to the accessible scope of the expression. It is also possible to read the status of a device using the dot '.' operator.

4.7.3 Arithmetic operators

4.7.3.1 Unary operators

- ! : boolean not
- ~ : bitwise negation
- + : unary plus
- - : unary -

4.7.3.2 Binary operators

- + : addition
- - : substraction
- / : division
- % : modulo
- * : multiply
- >> : right shift
- >>> : unsigned right shift
- << : left shift
- & : bitwise and
- | : bitwise and
- ^ : bitwise xor

- && : boolean and
- || : boolean or
- > : greater than
- < : lower than
- >= : greater or equal than
- <= : lower or equal than
- == : equals
- != : not equals
- #<operator> : string operations (may be extended through engine API)

4.7.3.3 Operators precedence

1) #
2) !, ~
3) unary +, -
4) *, /, %
5) binary +, -
6) <<, >>>, >>
7) <=, >=, <, >
8) ==, !=
9) &
10) ^
11) |
12) &&
13) ||

4.7.4 String operators

HomeEZ offers an extendable string operator "#" that allows to perform string operations. The list of operands must be provided between parenthesis, in a comma separated list of values.

The following operations are available by default:

- #equals(op1, op2, …) : returns true if operands are equal to each other, else returns false.
- #concat(op1, op2, ...) : concatenates the operands

- #trim(op) : returns the string with leading and trailing whitespace omitted
- #upper(op) : returns the string with all characters converted to upper case
- #lower(op) : returns the string with all characters converted to lower case
- #length(op) : returns the length of the string

4.7.5 Status access operator

Status access operator '.' allows to read the status of a device. It is only possible to read the status of a device with this operator (and not a group) as no merge method may be defined. The <status> mark may be used inside scenario statement for more advanced status fields manipulation (see §4.5.4.4).

5 APPENDIX

5.1 Uses cases examples

Next section provides readers with small samples of typical use cases.

5.1.1 Lounge heating control

A user wants to control her lounge heating by providing a threshold temperature. She has at her disposal two devices: (1) a temperature sensor and (2) a heater.

```
<homeez>
       <!-- creates a scenario that runs every 5 minutes from now -->
       <scenario name="LoungeHeatingControl">
              <body>
                 <invoke device="tempSensor" action="getTemperature" result="temp"/>
                 <if condition="temp &lt; 20">
                     <then>
                            <invoke device="heater" action="on" />
                     </then>
                     <else>
                            <invoke device="heater" action="off" />
                     </else>
                 </if>
              </body>

              <trigger>
                     <timer delay="5*60*1000" />
              </trigger>
       </scenario>
</homeez>
```

If asked for its pretty-print, the above scenario will display: "*Starting right now, and every 5 minutes, runs LoungeHeatingControl. Gets temperature and set heater "on" if temperature lower than 20.*"

5.1.2 Turn on and turn off light on presence detection

A user wants to turn on the light of its corridor on presence detection, and turn it off back after no presence is detected during 10 minutes.

```
<homeez>
  <scenario>
       <body>
         <monitor value="kill" scenario="TimeOff" />
         <invoke device="Light" action="on" />
         <monitor value="schedule" delay="10*60*1000" scenario="TimeOff" />
       </body>

       <trigger>
         <event device="Detector" name="presence"/>
       </trigger>
  </scenario>

  <scenario name="TimeOff">
    <body>
       <invoke device="Light" action="off" />
    </body>
  </scenario>
</homeez>
```

If asked for its pretty-pint, the first above scenario will display: "*When the Detector broadcasts the "presence" event, stop any "TimeOff" scenario, then if the "Light.onOff == 0" then the Light will on, then execute in 10mn the TimeOff scenario*".

5.1.3 Turn on corridor and hall lights on lounge switch on

A user wants to turn on both the corridor and the hall lights when the lounge switch is set to on.

```
<homeez>
       <!-- Sum up all lights together -->
       <configuration>
              <device name="LightNearTheDoor" includedIn="AllLights" />
              <device name="LightNearTheWindow" includedIn="AllLights" />
              <device name="LightEntranceCorridor" includedIn="AllLights" />
              <device name="LightEndCorridor" includedIn="AllLights" />
       </configuration>

       <scenario>
              <body>
                     <invoke device="AllLights" action="on" />
              </body>
              <trigger>
                     <event device="SwitchNearTheDoor" name="on" />
              </trigger>
       </scenario>
</homeez>
```

If asked for its pretty-print, the above scenario will display: "*When the SwitchNearTheDoor broadcasts "on" event, the Lights will "on"*".

5.1.4 Turn on TV in the launch area

A hotel wants to switch on the TV every day at 08:00. It can be turned off manually by employees etc... but in case of the detection of a presence, the hotel wants the TV to be switched on again.

```
<homeez>
       <scenarioClass name="samples.LaunchTV">
              <param name="channel" />
              <param name="detector" />
              <param name="tv" />
              <body>
                     <invoke action="on" device="tv">
                            <arg name="channel" value="channel" />
                     </invoke>
              </body>
       </scenarioClass>
       <scenario class="samples.LaunchTV">
              <arg name="channel" value="1" />
              <arg name="detector" value="LoungePresenceDetector" />
              <arg name="tv" value="TVLounge" />

              <trigger>
                     <event device="LoungePresenceDetector" name="active" />
                     <timer date="8:00:00" />
              </trigger>
       </scenario>
</homeez>
```

If asked for its pretty-print, the above scenario will display: "*When the LoungePresenceDetector broadcasts the "active" event, execute a LaunchTV scenario with TVLounge, LoungePresenceDetector, and 2 . Schedule this scenario every day at 08:00*".

5.1.5 Alert on bad identification

Every user has an electronic badge to get in. Some badges are out of date (the user has forgotten to pay his monthly bill). He can access the room / the service, but an alarm is sent to the system manager to recall the user to pay the bill.

```
<homeez>
	<scenario>
		<body>
			<invoke device="RFIDreaderAtTheDoor" action="getUser" result="user" />
			<invoke device="DBOfBills" action="hasPaidHisMonthlyBill"
				result="ok">
				<arg name="username" value="user" />
			</invoke>

			<if condition="!ok">
				<then>
					<invoke device="GSMmodule" action="sendSMS">
						<arg
							name="phoneNumber"
							value="&quot;+33699816268&quot;"
						/>
						<arg
					name="msg"
					value='#concat user " needs to pay its bill !"'
						/>
					</invoke>
				</then>
			</if>
			<invoke device="TheDoor" action="unlock" />
		</body>

		<trigger>
			<event device="RFIDreaderAtTheDoor" name="in" />
		</trigger>
	</scenario>
</homeez>
```

If asked for its pretty-print, the above scenario will display : "*When the RFIDreaderAtTheDoor broadcasts "in" event, if t RFIDreaderAtTheDoor.date < Home.date then an SMS is sent with the message "[RFIDreaderAtTheDoor.name] needs to pay its bill !*".

5.1.6 Complex temperature device

The user has two temperature sensors, one inside, and one outside. The threshold temperature for some regulation scenario is made of the two sensors, thanks to an "added value formula".
This scenario is written using a generic regulation scenario provided by a third party. Using it only requires providing the names of the devices actually paired on the system:

```
<homeez>
	<configuration>
		<!-- Declare a virtual device that holds the target temp -->
		<device name="TargetTemp">
			<status name="threshold" value="20" />
		</device>
	</configuration>

	<!-- creates a scenario that runs the samples.HeatingControl class -->
	<scenario class="samples.HeatingControl" name="LoungeRegul">
		<arg name="tempIn" value="SensorTemp" />
		<arg name="tempOut" value="SensorTemp" />
		<arg name="heater" value="HeaterNearTheDoor" />
		<arg name="target" value="TargetTemp" />
	</scenario>
</homeez>
```

If asked for its pretty-print, the above scenario will display: "*Starting right now, runs a sample.HeatingControl with SensorTemp, HeaterNearTheDoor and TargtTemp.*"

The third party scenario class may be defined in the following way:

```
<homeez>
        <!-- a generic regulation every 5 minutes -->
        <scenarioClass name="samples.HeatingControl">
                <param name="tempIn" />
                <param name="tempOut" />
                <param name="heater" />
                <param name="target" />
                <body>
                        <invoke device="tempIn" action="getTemp" result="tIn" />
                        <invoke device="tempOut" action="getTemp" result="tOut" />
                        <expression value="(2*tIn + 4*tOut)/6" result="tmp" />
                        <if condition="tmp &#8249; target">
                                <then>
                                        <invoke device="heater" action="on" />
                                </then>
                                <else>
                                        <invoke device="heater" action="off" />
                                </else>
                        </if>
                </body>

                <trigger>
                        <timer value="0" every="5*60*1000" />
                </trigger>
        </scenarioClass>
</homeez>
```

If asked for its pretty-print, the above scenario will display: “E*very 5 minutes, runs a HeatingControl with tempIn, tempOut, heater and target.*”

5.2 XSD File

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://e-s-r.net/homeez/xml" xmlns="http://e-s-r.net/homeez/xml"
      elementFormDefault="qualified">
      <xsd:element name="homeez">
            <xsd:complexType>
                <xsd:choice minOccurs="0" maxOccurs="unbounded">
                  <!-- element CONFIGURATION -->
                  <xsd:element name="configuration" type="configurationType" minOccurs="0" maxOccurs="1">
                              <xsd:unique name="UniqueConfigurationDeviceName">
                                    <xsd:selector xpath="device" />
                                    <xsd:field xpath="@name" />
                              </xsd:unique>
                              <xsd:unique name="UniqueConfigurationDeviceUid">
                                    <xsd:selector xpath="device" />
                                    <xsd:field xpath="@uid" />
                              </xsd:unique>
                        </xsd:element>

                        <!--element SCENARIO CLASS -->
                        <xsd:element name="scenarioClass" type="scenarioClassType" minOccurs="0" maxOccurs="unbounded">
                        </xsd:element>

                        <!--element SCENARIO -->
                        <xsd:element name="scenario" type="scenarioType" minOccurs="0" maxOccurs="unbounded">
                        </xsd:element>
                </xsd:choice>

                  <xsd:attribute name="name" type="xsd:string" use="optional" />
                  <xsd:attribute name="version" type="xsd:string" use="optional" />
            </xsd:complexType>

            <!-- DECLARATION OF HOMEEZ ELEMENT UNIQUE ATTRIBUTE -->
            <xsd:unique name="UniqueHomeeZDeviceName">
                  <xsd:selector xpath="device" />
                  <xsd:field xpath="@name" />
            </xsd:unique>
            <xsd:unique name="UniqueHomeeZDeviceUid">
```

```
                <xsd:selector xpath="device" />
                <xsd:field xpath="@uid" />
            </xsd:unique>
        </xsd:element>
        <!-- END element HOME-EZ -->

        <!-- DECLARATION OF TYPE CONFIGURATION -->
        <xsd:complexType name="configurationType">
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="device" type="deviceType"/>
                <xsd:element name="group" type="groupType"/>
            </xsd:choice>
        </xsd:complexType>

        <!-- DECLARATION OF TYPE DEVICE -->
        <xsd:complexType name="deviceType">
            <xsd:sequence>
                <xsd:element name="action" type="actionType" minOccurs="0" maxOccurs="unbounded" />
                <xsd:element name="status" type="statusType" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="group" type="groupType" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="deviceName" use="required"/>
            <xsd:attribute name="includedIn" type="deviceName" use="optional"/>
            <xsd:attribute name="uid" type="xsd:string" use="optional"/>
        </xsd:complexType>

        <!-- DECLARATION OF TYPE DEVICE NAME -->
        <xsd:simpleType name="deviceName">
            <xsd:restriction base="identifier"/>
        </xsd:simpleType>

        <!-- DECLARATION OF TYPE SCENARIO CLASS -->
        <xsd:complexType name="scenarioClassType">
            <xsd:sequence>
                <xsd:element name="param" type="paramType" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="body" minOccurs="1" maxOccurs="1" type="bodyType"/>
                <xsd:element name="trigger" type="triggerType" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <!-- This mandatory attribute represents the name of the scenario class. -->
            <xsd:attribute name="name" type="fullyQualifiedName" use="required"/>
```

```
	</xsd:complexType>

	<!--DECLARATION OF TYPE SCENARIO -->
	<xsd:complexType name="scenarioType">
		<xsd:sequence>
			<xsd:element name="arg" type="argType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="body" minOccurs="0" maxOccurs="1" type="bodyType"/>
			<xsd:element name="trigger" type="triggerType" minOccurs="0" maxOccurs="1" />
		</xsd:sequence>
		<xsd:attribute name="name" type="fullyQualifiedName" use="optional" />
		<xsd:attribute name="class" type="fullyQualifiedName" use="optional" />
		<xsd:attribute name="initialState" type="scenarioStateValue" use="optional" />
	</xsd:complexType>

	<!-- DECLARATION OF TYPE INITIAL STATE VALUE -->
	<xsd:simpleType name="scenarioStateValue">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="init"/>
			<xsd:enumeration value="active"/>
			<xsd:enumeration value="inactive"/>
			<xsd:enumeration value="queued"/>
		</xsd:restriction>
	</xsd:simpleType>

	<!-- DECLARATION OF TYPE BODY -->
	<xsd:complexType name="bodyType">
		<xsd:choice minOccurs="0" maxOccurs="unbounded">
			<xsd:element name="monitor" type="monitorType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="kill" type="killType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="schedule" type="scheduleType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="resume" type="resumeType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="suspend" type="suspendType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="if" type="ifType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="invoke" type="invokeType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="status" type="statusStatementType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="expression" type="expressionStatementType" minOccurs="0" maxOccurs="unbounded"/>
		</xsd:choice>
	</xsd:complexType>

	<!-- DECLARATION OF TYPE ACTION -->
```

```
      <xsd:complexType name="actionType">
            <xsd:sequence>
                  <xsd:element name="param" type="paramType" minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
            <xsd:attribute name="name" type="actionName" use="required" />
      </xsd:complexType>

      <!-- DECLARATION OF TYPE ACTION NAME -->
      <xsd:simpleType name="actionName">
            <xsd:restriction base="identifier"/>
      </xsd:simpleType>

      <!-- DECLARATION OF TYPE STATUS NAME -->
      <xsd:simpleType name="statusName">
            <xsd:restriction base="identifier"/>
      </xsd:simpleType>

      <!-- DECLARATION OF TYPE GROUP -->
      <xsd:complexType name="groupType">
            <xsd:attribute name="name" type="deviceName" use="required"/>
            <xsd:attribute name="parent" type="deviceName" use="optional"/>
      </xsd:complexType>

      <!-- DECLARATION OF TYPE STATUS -->
      <xsd:complexType name="statusType">
            <xsd:attribute name="name" type="identifier" use="required"/>
            <xsd:attribute name="value" type="expression" use="required"/>
      </xsd:complexType>

      <!-- DECLARATION OF TYPE PARAM -->
      <xsd:complexType name="paramType">
            <xsd:attribute name="name" type="identifier" use="required" />
      </xsd:complexType>

      <!-- DECLARATION OF TYPE ARG TYPE -->
      <xsd:complexType name="argType">
            <xsd:complexContent>
                  <xsd:extension base="paramType">
                        <xsd:attribute name="value" type="expression" use="required" />
                  </xsd:extension>
```

```
        </xsd:complexContent>
    </xsd:complexType>

    <!-- DECLARATION OF TYPE MONITOR BASE TYPE -->
    <xsd:complexType name="monitorBaseType">
        <xsd:complexContent>
            <xsd:extension base="statementType">
                <xsd:attribute name="scenario" type="fullyQualifiedName" use="required" />
                <xsd:attribute name="delay" type="expression" use="optional" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- DECLARATION OF TYPE MONITOR TYPE -->
    <xsd:complexType name="monitorType">
        <xsd:complexContent>
            <xsd:extension base="monitorBaseType">
                <xsd:attribute name="value" type="monitorValueType" use="required" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- DECLARATION OF TYPE MONITOR VALUE -->
    <xsd:simpleType name="monitorValueType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="kill" />
            <xsd:enumeration value="schedule" />
            <xsd:enumeration value="resume" />
            <xsd:enumeration value="suspend" />
        </xsd:restriction>
    </xsd:simpleType>

    <!-- DECLARATION OF TYPE KILL TYPE -->
    <xsd:complexType name="killType">
        <xsd:complexContent>
            <xsd:extension base="monitorBaseType"/>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- DECLARATION OF TYPE SCHEDULE TYPE -->
```

```
      <xsd:complexType name="scheduleType">
            <xsd:complexContent>
                  <xsd:extension base="monitorBaseType"/>
            </xsd:complexContent>
      </xsd:complexType>

      <!-- DECLARATION OF TYPE RESUME TYPE -->
      <xsd:complexType name="resumeType">
            <xsd:complexContent>
                  <xsd:extension base="monitorBaseType"/>
            </xsd:complexContent>
      </xsd:complexType>

      <!-- DECLARATION OF TYPE SUSPEND TYPE -->
      <xsd:complexType name="suspendType">
            <xsd:complexContent>
                  <xsd:extension base="monitorBaseType"/>
            </xsd:complexContent>
      </xsd:complexType>

      <!-- DECLARATION OF TYPE MERGE -->
      <xsd:simpleType name="mergeType">
            <xsd:restriction base="identifier">
            </xsd:restriction>
      </xsd:simpleType>


      <!-- DECLARATION OF TYPE ABSTRACT STATEMENT -->
      <xsd:complexType name="abstractStatementType">
      </xsd:complexType>

      <!-- DECLARATION OF TYPE STATEMENT -->
      <xsd:complexType name="statementType">
            <xsd:complexContent>
                  <xsd:extension base="abstractStatementType">
                  </xsd:extension>
            </xsd:complexContent>
      </xsd:complexType>

      <!-- DECLARATION OF TYPE IF -->
```

```
<xsd:complexType name="ifType">
      <xsd:complexContent>
            <xsd:extension base="statementType">
                  <xsd:sequence>
                        <xsd:element name="then" minOccurs="1" maxOccurs="1" type="bodyType" />
                        <xsd:element name="else" minOccurs="0" maxOccurs="1" type="bodyType" />
                  </xsd:sequence>
                  <xsd:attribute name="condition" type="expression" use="required" />
            </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>


<!-- DECLARATION OF TYPE TRIGGER -->
<xsd:complexType name="triggerType">
      <xsd:complexContent>
            <xsd:extension base="statementType">
                  <xsd:sequence>
                        <xsd:choice minOccurs="0" maxOccurs="unbounded">
                              <xsd:element name="event" type="eventType" minOccurs="0" maxOccurs="unbounded" />
                              <xsd:element name="timer" type="timerType" minOccurs="0" maxOccurs="unbounded" />
                        </xsd:choice>
                  </xsd:sequence>
            </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>

<!-- DECLARATION OF TYPE INVOKE -->
<xsd:complexType name="invokeType">
      <xsd:complexContent>
            <xsd:extension base="statementType">
                  <xsd:sequence>
                        <xsd:element name="arg" type="argType" minOccurs="0"
                              maxOccurs="unbounded" />
                  </xsd:sequence>
                  <xsd:attribute name="action" type="actionName" use="required" />
                  <xsd:attribute name="device" type="deviceName" use="required" />
                  <xsd:attribute name="onError" type="xsd:string" use="optional" />
                  <xsd:attribute name="result" type="identifier" use="optional" />
                  <xsd:attribute name="merge" type="mergeType" use="optional" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- DECLARATION OF TYPE STATUS -->
    <xsd:complexType name="statusStatementType">
        <xsd:complexContent>
            <xsd:extension base="statementType">
                <xsd:sequence>
                    <xsd:element name="status" type="statusStatementType" minOccurs="0"   maxOccurs="unbounded" />
                </xsd:sequence>
                <xsd:attribute name="device" type="deviceName" use="required" />
                <xsd:attribute name="name" type="statusName" use="required" />
                <xsd:attribute name="value" type="expression" use="optional" />
                <xsd:attribute name="result" type="identifier" use="optional" />
                <xsd:attribute name="merge" type="mergeType" use="optional" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="timerType">
        <xsd:complexContent>
            <xsd:extension base="triggerStatementType">
                <xsd:attribute name="startAt" type="timerPeriodicDate" use="optional" />
                <xsd:attribute name="endAt" type="timerPeriodicDate" use="optional" />
                <xsd:attribute name="date" type="timerPeriodicDate" use="optional" />
                <xsd:attribute name="delay" type="expression" use="optional" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:simpleType name="timerPeriodicDate">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="[0-9d :*-/,]*"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="eventType">
        <xsd:complexContent>
            <xsd:extension base="triggerStatementType">
```

```
                        <xsd:attribute name="device" type="deviceName" use="required" />
                        <xsd:attribute name="name" type="eventName" use="required" />
                </xsd:extension>
            </xsd:complexContent>
    </xsd:complexType>

    <xsd:simpleType name="eventName">
            <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>

    <xsd:complexType name="triggerStatementType">
            <xsd:complexContent>
                <xsd:extension base="abstractStatementType">
                </xsd:extension>
            </xsd:complexContent>

    </xsd:complexType>

    <!-- DECLARATION OF TYPE EXPRESSION STATEMENT -->
    <xsd:complexType name="expressionStatementType">
            <xsd:complexContent>
                <xsd:extension base="statementType">
                        <xsd:attribute name="value" type="expression" use="required" />
                        <xsd:attribute name="result" type="identifier" use="required" />
                </xsd:extension>
            </xsd:complexContent>
    </xsd:complexType>

    <!-- DECLARATION OF TYPE IDENTIFIER -->
    <xsd:simpleType name="identifier">
            <xsd:restriction base="xsd:string">
                <xsd:pattern value="[a-zA-Z_0-9][a-zA-Z_0-9]*"/>
            </xsd:restriction>
    </xsd:simpleType>

    <!-- DECLARATION OF TYPE FULLY QUALIFIED NAME -->
    <xsd:simpleType name="fullyQualifiedName">
            <xsd:restriction base="xsd:string">
                <xsd:pattern value="[a-zA-Z_0-9][a-zA-Z_0-9]*([.][a-zA-Z_0-9][a-zA-Z_0-9]*)*"/>
            </xsd:restriction>
```

```
      </xsd:simpleType>

      <!-- DECLARATION OF TYPE EXPRESSION -->
      <xsd:simpleType name="expression">
            <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>

</xsd:schema>
```