YAHAS

Ruleset

Sellitto Nicola

v0.2 – 22/03/2020

Sommario

1. Generalità

Di seguito sono riportate le specifiche del mini linguaggio di script per la definizione delle rules.

L'engine di valutazione del ruleset è attivato dal servzio YManager della componente NodeManager.

2. Ruleset

Il ruleset definisce gli oggetti del sistema e la logica di funzionamento ad eventi; tutte le definizioni sono indicate in sezioni distinte.

2.1. Sezioni

Il file ruleset è composto da 8 sezioni indipendenti, sempre presenti in ordine prestabilito.

La section inizia con il proprio token identificativo e termina in presenza del token END.

Una section può essere vuota non avendo definizioni al suo interno ma deve essere sempre dichiarata.

Le section ed il loro ordine risultano:

- FLOAT:
- INTEGER:
- STRING:
- ACTUATOR:
- SENSOR:
- TIMER:
- TASK:
- RULE:

2.2. Variabili

Nelle prime 3 sezioni si definiscono le variabili utilizzate, queste risultano tutte globali ed appartenenti ai seguenti 3 tipi:

- Float
- Integer
- String

La definizione di una variabile implica anche la loro valorizzazione iniziale; per le variabili di tipo Float il carattere separatore della parte decimale è “.” il punto .

I caratteri permessi nella nomenclatura delle variabili risultano:

- alfabetici senza distinzione del case
- numerici
- speciale _

Il nome di una variabile inizia sempre con una lettera, inoltre Il valore delle variabili string è delimitato agli estrimi da carattere ".
Di seguito si riportano esempi di definizione:

```
FLOAT:
      Temp = 20.3
END


INTEGER:
      tot_Squilli = 0
END


STRING:
    init = "true"
    oldState = "OFF"
END
```

2.3. Devices

Nelle successive 3 sezioni si definiscono i devices refenenziati nelle rules; la definizione si limita alla sola dichiarazione del nome del device e non ai suoi attributi che devono essere stati precedentemente definiti nel db json di configurazione.

Il tipo di devices dichiarabile sono:

- ACTUATOR
- SENSOR
- TIMER

Il nome del device è quello identificativo (full path comprendente del nome agent di appartenenza) e non quello descrittivo.

Di seguito si riportano esempi di definizione:

```
ACTUATOR:
    name    shellyone/campanello
    name    sonoffmini/tvsony
END


SENSOR:
    name    sonoffmini/termostato
END


TIMER:
    name    manager/sveglia
END
```

I Timer indicati nelle rules sono quelli globali appartenenti al NodeManager e non quelli locali appartenenti ai singoli NodeAgent.

2.4. Task

Nella sezione successiva dei Task si definiscono le subroutine costituite da un insieme ordinato di Action ognuna implementata da uno specifico statement.

La sezione può contenere multiple istanze di task distinti, ognuno inizia con l'attributo del NAME, e termina con il token END.

All'interno del singolo task sono definite le varie ACTION da eseguire.

L'esecuzione di un Task avviene quando viene esplicitamente richiamato dallo statement CALL.

Un task **non** può richiamare un altro Task.

Le Action hanno il seguente formato sintattico:

```
<Action> :=  <Id> <Type> <Name> <Attribute> <Value>
```

Il token ID identifica lo statement dell'Action che può essere: `SET, GET, CALL`

2.4.1. SET

Lo statement SET imposta il valore dell'attributo dell'oggetto referenziato, alcuni attributi hanno valori predefiniti altri valori user defined.

Prima del nome dell'oggetto bisogna sempre indicarne il tipo.

Di seguito si riporta lo schema generale:

<Id>	<Type>	<Name>	<Attribute>	<Value>
SET	ACTUATOR	text value	MODE	AUTOMATED | MANUAL
			OPERATION	ENABLED | DISABLED
			STATE	ON | OFF
			COUNT	int Exp
	SENSOR	text value	MODE	AUTOMATED | MANUAL
			OPERATION	ENABLED | DISABLED
	TIMER	text value	OPERATION	ENABLED | DISABLED
			WAKEUP	ON | OFF
	FLOAT	text value	VALUE	float Exp
	INTEGER	text value	VALUE	int Exp
	STRING	text value	VALUE	constant | var string

Gli attributi ed i loro valori predefiniti risultano:

```
MODE        :=  AUTOMATED   |  MANUAL
OPERATION   :=  DISABLED    |  ENABLED
STATE       :=  OFF   |  ON
WAKEUP      :=  OFF   |  ON
```

Gli attributi con valori predefiniti sono tutti del tipo stringa ma tali valori essendo predefiniti non devono essere delimitati da “.

L’impostazione degli attributi riferiti a Sensor & Actuator si traduce in una richiesta inviata allo specifico device a cui corrisponde l’attesa da parte del device stesso di avvenuta impostazione.

Pertanto l’esecuzione del SET per Sensor & Actuator non garantisce l’azione; il timeout di avvenuta impostazione è di 2 secondi.

Di seguito si riportano alcuni esempi:

```
SET   STRING      init                    VALUE       "false"
SET   INTEGER     totSquilli              VALUE       totSquilli+1
SET   ACTUATOR    sonoffmini/tvsony       STATE       ON
```

2.4.2. GET

Lo statement GET legge il valore dell’attributo per l’oggetto referenziato valorizzando una variabile.

Di seguito si riporta lo schema generale:

<Id>	<Type>	<Name>	<Attribute>	<Value>
GET	ACTUATOR	text value	MODE	varName
			OPERATION	
			STATE	
			COUNT	
	SENSOR	text value	MODE	varName
			OPERATION	
			HUMIDITY	
			TEMPERATURE	
			MEASURE	
	TIMER	text value	OPERATION	varName
			WAKEUP	

La variabile di destinazione deve essere dello stesso tipo dell'attributo referenziato.

Di seguito si riportano alcuni esempi:

```
GET   ACTUATOR    sonoffmini/tvsony       STATE         tvstato
GET   SENSOR      shellyone/termometro    TEMPERATURA   tempvalue
```

2.4.3. CALL

Lo statement CALL permette di eseguire Task o Function predefinite.

Di seguito si riporta lo schema generale:

<Id>	<Type>	<Name>	<Attribute>	<Value>
CALL	FUNCTION	text value	ARG	value
	FUNCTION	text value		
	TASK	text value		

Poiché tutte le variabili del ruleset sono globali ed essendo i Task definiti dall'utente non è necessario passare parametri ai Task.

Il passaggio ed il ritorno di valori per le Function (che sono tutte predefiniti nel sistema) può avvenire in 2 modalità.

Nella modalità diretta si utilizza l'attributo ARG specificando la variabile considerata come argument.

Nella modalità indiretta si utilizza la forma senza indicare l'attributo ARG, in questo caso la funzione preleva i suoi argument dalle variabili di sistema predefinite che risultano:

```
fArg1, fARg2, fARg3          per le Float
iArg1, iARg2, iARg3          per gli Integer
sArg1, sARg2, sARg3          per le String
```

Ad esempio la funzione delayMs può essere richiamata nelle 2 modalità seguenti:

```
CALL   FUNCTION    delayMs          ARG        10
SET    INTEGER     tempo            VALUE      20
CALL   FUNCTION    delayMs          ARG        tempo

SET    INTEGER     iArg1            VALUE      30
CALL   FUNCTION    delayMs
```

Eventuali valori ritornati dalle funzioni sono sempre impostati nelle seguenti variabili di sistema:

```
fRet1, fRet2, fRet3          per le Float
iRet1, iRet2, iRet3          per gli Integer
sRet1, sRet2, sRet3          per le String
```

E' importante l'ordine delle variabili di sistema, sia xArg che xRet sono utilizzate automaticamente in sequenza dalle funzioni.

Di seguito si riporta un esempio completo di Task:

```
TASK:

      NAME  task1
            SET   INTEGER     total                   VALUE         7
            GET   SENSOR      shellyone/termometro    TEMPERATURA   tempvalue
            CALL  FUNCTION    delayMs                 VALUE         100
      END
END
```

2.5. Rule

L'ultima sezione è quella della RULE dove si esaminano gli eventi per intraprendere specifiche azioni.

La section RULE non può mai essere vuota.

La sezione può contenere multiple istanze di regole distinte, ognuna inizia con l'attributo del NAME, e termina con il token END.

All'interno della singola regola sono definite prima le varie CONDITION, per esaminare gli eventi, successivamente si indica l'espressione logica da valutare, infine segue la definizione delle ACTION da eseguire se l'espressione logica risulta soddisfatta.

2.5.1. Condition

Le Condition rappresentano gli operandi dell'espressione booleana da valutare.

Le Condition hanno il seguente formato sintattico:

```
<Condition> := <Id> <Type> <Name> <Attribute> <Operator> <Value>
```

Il token ID identifica il singolo operando e sarà utilizzato nella definizione dell'espressione boolean.

Il token ID è composto dal prefisso `OP_` seguito da una singola lettera sequenziale partendo dalla A.

Di seguito si riporta lo schema generale:

CONDITION					
<Id>	**<Type>**	**<Name>**	**<Attribute>**	**<Operator>**	**<Value>**
OP_x	ACTUATOR	text value	MODE	EQ | NE	AUTOMATED | MANUAL
			OPERATION	EQ | NE	ENABLED | DISABLED
			STATE	EQ | NE	ON | OFF
			COUNT	EQ .. NE	int Exp
	SENSOR	text value	MODE	EQ | NE	AUTOMATED | MANUAL
			OPERATION	EQ | NE	ENABLED | DISABLED
			HUMIDITY	EQ .. NE	int Exp
			TEMPERATURE	EQ .. NE	float Exp
			MEASURE	EQ .. NE	int Exp
	TIMER	text value	OPERATION	EQ | NE	ENABLED | DISABLED

			WAKEUP	EQ | NE	ON | OFF
	FLOAT	text value	VALUE	EQ .. NE	float Exp
	INTEGER	text value	VALUE	EQ .. NE	int Exp
	STRING	text value	VALUE	EQ .. NE	constat|var string

L'espressione logica si definisce mediante lo statement :

LEXP	<Logical_Expression>

Esempio:

```
RULE:

      NAME  prova
            OP_A  INTEGER          total                   value EQ    3
            OP_B  ACTUATOR         wemosmini/pulsante      state EQ    OFF
            LEXP  A & B
            SET   ACTUATOR         wemosmini/pulsante      state EQ    ON
            SET   INTEGER          total                   value EQ    0
      END
END
```

2.5.2. Action

Le Action hanno il seguente formato sintattico:

<Action> := <Id> <Type> <Name> <Attribute> <Value>

Il token ID identifica lo statement dell'Action che può essere: `SET, GET, CALL`

Le Action delle Rule sono le stesse di quelle dei Task pertanto si rimanda al paragrafo Task.Action per ulteriori dettagli.

3. Esempio

Di seguito è riportato un esempio completo di utilizzo dei vari statement.

Scopo: aprire un cancello dopo 5 secondi dalla ricezione del terzo squillo di un campanello.

Si ipotizza di usare i 2 attuatori: un pulsante per campanello ed un interruttor per cancello.
Tralasciando i dettagli di configurazione degli attuatori si ipotizza che essi siano gestiti il primo da una board shellyone e l'altro da un sonoffmini.

```
// ------------------------
// FLOAT SECTION
// ------------------------
FLOAT:
END

// ------------------------
// INTEGER SECTION
// ------------------------
INTEGER:
    totsquilli=0
END

// ------------------------
// STRING SECTION
// ------------------------
STRING:
    init="true"
    oldstate="OFF"
END

// ------------------------
// ACTUATOR SECTION
// ------------------------
ACTUATOR:
    NAME    sonoffmini/cancello
    NAME    shellyone/campanello
END

// ------------------------
// SENSOR SECTION
// ------------------------
SENSOR:
END

// ------------------------
// TIMER SECTION
// ------------------------
TIMER:
END

// ------------------------
// TASK SECTION
// ------------------------
TASK:

/*
Task di inizializzazione sistema, esempio di Task eseguito una volta
*/
    NAME    Startup
        CALL    FUNCTION    print                   ARG         "Hello World"
    END
```

```
/*
Attivazione cancello
*/
    NAME    ApriCancello
        CALL    FUNCTION    print                   ARG         "Aspetto 5 sec"
        CALL    FUNCTION    delaySec                ARG         5
        SET     ACTUATOR    sonoffmini/cancello     STATE       on
        CALL    FUNCTION    print                   ARG         "Cancello aperto"
    END

END

// ------------------------
// RULE SECTION
// ------------------------
RULE:

/*
Rule di inizializzazione sistema, viene eseguito soltanto una volta
*/
    NAME    CheckStartUp
        OP_A    STRING      init                    VALUE       EQ  "true"
        LEXP    (A)
        SET     STRING      init                    VALUE       "false"
        CALL    TASK        startup
    END

/*
Conta ogni cambio stato OFF -> ON
*/
    NAME    ContaSquilli
        OP_A    ACTUATOR    shellyone/campanello    STATE       EQ  on
        OP_B    STRING      oldstate                VALUE       EQ  "OFF"
        LEXP    (A&B)
        SET     STRING      oldstate                VALUE       "ON"
        CALL    FUNCTION    print                   ARG         "New Pulse"
        SET     INTEGER     totsquilli              VALUE       totsquilli+1
    END

/*
Ricorda nuovo cambio stato ON -> OFF
*/
    NAME    SetOff
        OP_A    ACTUATOR    shellyone/campanello    STATE       EQ  off
        OP_B    STRING      oldstate                VALUE       EQ  "ON"
        LEXP    (A&B)
        SET     STRING      oldstate                VALUE       "OFF"
    END

/*
 * Verifica il numero di squilli,
 * al 3 squillo attende 5 secondi poi apre il cancello
*/
    NAME    CheckSquilli
        OP_A    INTEGER     totsquilli              VALUE       EQ  3
        LEXP    (A)
        SET     INTEGER     totsquilli              VALUE       0
        CALL    FUNCTION    print                   ARG         "Ricevuti 3 squilli"
        CALL    TASK        apricancello
    END

END
```

4. Appendice

4.1. Action

ACTION				
<Id>	**<Type>**	**<Name>**	**<Attribute>**	**<Value>**
SET	ACTUATOR	text value	MODE	AUTOMATED | MANUAL
			OPERATION	ENABLED | DISABLED
			STATE	ON | OFF
			COUNT	int Exp
	SENSOR	text value	MODE	AUTOMATED | MANUAL
			OPERATION	ENABLED | DISABLED
	TIMER	text value	OPERATION	ENABLED | DISABLED
			WAKEUP	ON | OFF
	FLOAT	text value	VALUE	float Exp
	INTEGER	text value	VALUE	int Exp
	STRING	text value	VALUE	constat|var string
GET	ACTUATOR	text value	MODE	varName
			OPERATION	
			STATE	
			COUNT	
	SENSOR	text value	MODE	varName
			OPERATION	
			HUMIDITY	
			TEMPERATURE	
			MEASURE	
	TIMER	text value	OPERATION	varName
			WAKEUP	
CALL	FUNCTION	text value	ARGS	value
	FUNCTION	text value		
	TASK	text value		

4.2. Condition

CONDITION					
<Id>	**<Type>**	**<Name>**	**<Attribute>**	**<Operator>**	**<Value>**
OP_x	ACTUATOR	text value	MODE	EQ | NE	AUTOMATED | MANUAL
			OPERATION	EQ | NE	ENABLED | DISABLED
			STATE	EQ | NE	ON | OFF
			COUNT	EQ .. NE	int Exp
	SENSOR	text value	MODE	EQ | NE	AUTOMATED | MANUAL
			OPERATION	EQ | NE	ENABLED | DISABLED
			HUMIDITY	EQ .. NE	int Exp
			TEMPERATURE	EQ .. NE	float Exp
			MEASURE	EQ .. NE	int Exp
	TIMER	text value	OPERATION	EQ | NE	ENABLED | DISABLED
			WAKEUP	EQ | NE	ON | OFF
	FLOAT	text value	VALUE	EQ .. NE	float Exp
	INTEGER	text value	VALUE	EQ .. NE	int Exp
	STRING	text value	VALUE	EQ .. NE	constat|var string

LEXP	<Logical_Expression>

4.3. Function

Name	Argument	Return
debug	ON | OFF	
delayMs	costant | var integer	
delaySec	costant | var integer	
floatToString	var float	sRet1
getWeekDay	tbd	
intToString	var integer	iRet1
print	costant | var string	
sendMail	tbd	
getTime		iRet1
resetTime		