

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Introduction

Program Logic Event Generator (PLEG) is a powerful and flexible tool for initiating **Actions** on the basis of the state of one or more **Inputs**. Inputs may be logical states (**Triggers**), variable values (**Device Properties**) or based on time (**Schedules**). The **Conditions** under which the **Actions** will be executed are described by expressions that can include logical, arithmetic, string or time-sequence terms. PLEG can be used for both simple and complex scenarios - the principles are the same.

PLEG has many advantages over standard Vera scenes (with or without Lua code). One of the most significant is that, following a Vera restart, PLEG will restart its Schedules accurately and process any changes that occurred during the restart.

PLEG is not difficult to use but it does require you to translate your requirements into logical expressions. This may require a little practice so it is often better to start with simple scenarios and work your way up. I recommend setting up a PLEG with a few *VirtualSwitch* devices so that you can test your logic before trying it out on your real devices. The *VariableContainer* plugin can also be used as a source or recipient of analogue values whilst you test your logic.

Forum members have asked how to use PLEG for almost every possible scenario and have rarely been disappointed. There is a very good chance that something close to your requirements has already been discussed so a thorough search using *Google* may pay dividends. Even with a pre-tested recipe, though, you will need to understand how it works. To help newcomers, I will attempt to explain in this guide the principles and practice of using PLEG.

The guide consists of the following sections: (Click the links to go to the page)

- [Names](#)
- [Triggers](#)
- [Device Properties](#)
- [Schedules](#)
- [Conditions](#)
- [Logical Expressions](#)
- [Numerical Comparisons](#)
- [Arithmetic Expressions](#)
- [Conditional Expressions](#)
- [Condition Values](#)
- [String Expressions](#)
- [Working with Time](#)
- [Sequence Expressions](#)
- [Multiple Trigger Expressions](#)
- [Special Conditions](#)
- [State Variables](#)
- [Actions](#)
- [Working with PLEG](#)
- [Licensing](#)
- [Examples](#)

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Names

Names of **Triggers**, **Device Properties**, **Schedules** and **Conditions** must be unique and may not start with a number or contain any spaces. Case is not significant: LightOn is the same as lighton or LIGHTON.

### Triggers

PLEG Triggers are references to logical events. They can be either true or false. The time when the state last changed is noted as the Trigger's *timestamp*.

Each device and plugin in Vera pre-defines a set of available Triggers. A Trigger may reflect the status of various elements of a device or plugin - depending on the type of device concerned. Examples include: *on/off*, *open/closed*, *tripped/not-tripped*. This type of Trigger may be set to reflect either sense of a status - a Trigger could be true when the device is turned off.

Triggers may also be associated with a level and a comparison. Examples include: *temperature goes-above 20*; *light-level goes-below 100*. In some devices, a Trigger may be set for a particular value. Example: *mode equals "Heat"*.

Create a PLEG trigger by selecting **Inputs** -> **Triggers** then selecting a **Device** from the pull-down list. The options in **Event Type** will depend on the type of device selected. Select the type of event for which this Trigger is required. Enter a **Name** for this Trigger. It will help later if this name has some obvious meaning - e.g. *BedLightOn*. Select **which state** to determine the sense of this Trigger or, if the Trigger is a comparison, enter the threshold value. Finally click on **Accept**.

It is not necessary to create Triggers for both senses of each status or comparison. You will see later that logical operators may be used to invert the sense of a Trigger in Condition expressions. The *timestamp* of a Trigger's false/true or true/false transition can also be included in Condition expressions by using the # or #! operators.

Name	Description	Actions
ItsNight	Day or Night indicates Night time.	Edit Delete
VS1On	VS1 is turned on	Edit Delete
TempLow	House Temperature ambient temperature goes below 15 degrees	Edit Delete
DimLights	IR code received	Edit Delete

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Device Properties

PLEG Device Properties may be used to access variables from devices or plugins. These will be represented as analogue values even if they reflect on/off states. The time when the value last changed is noted as the Device Property's *timestamp*.

PLEG will accept almost any type of variable as a Device Property. If the value is numeric, it may be used in expressions for calculations or numeric comparisons. String values may be compared to each other, or to constants, by the use of special operators. Example variables that may be used as a Device Property include: *CurrentSetpoint*, *LoadLevelStatus*, *LastUpdate*, *BatteryLevel*, *ModeStatus*.

Create a PLEG Device Property by selecting **Inputs -> Device Properties** then selecting a **Device** from the pull-down list. The options in the next pull-down box will depend on the type of device selected. Select the variable this Device Property should reflect. Click **Create**. To change the name of the new Device Property, click **Edit**, change the name from the default and click **Accept**.

Test

Room: Z\_Parked

Control Inputs Conditions Actions Advanced Logs Notifications

Triggers Schedules Device Properties

Close dialog and press SAVE to commit changes.

#58 House Heat Relay ModeStatus

Accept HWRMode

Name	Device Name	Device Variable	Current Value	Actions	
DayOfWeek	Away Lights Timer	DOW	2	Edit	Delete
HouseTemp	House Temperature	CurrentTemperature	19	Edit	Delete
LastAlarm	Alarm Tripped	LastTrip	1384892157	Edit	Delete
HWRMode	House Heat Relay	ModeStatus	Off	Edit	Delete

### Schedules

PLEG provides a comprehensive set of options for time Schedules. Each Schedule should have a **Start Type** and may optionally have a **Stop Type**. If there is no **Stop Type** specified, the Schedule will only be true for one cycle of PLEG evaluation. When both **Start Type** and **Stop Type** are specified, the Schedule will remain true throughout the inclusive period. The time when the Schedule last became true is noted as the Schedule's *timestamp*.

**Start Type** and **Stop Type** mostly provide the same options. The exceptions are noted below:

- **Interval** - Repetitively becomes true every hh:mm:ss. Will be pinned to exact times so each hour will be on the hour, etc.
- **Day of Week** - Selectable for any or all days of the week. The time is specified as shown below.
- **Day of Month** - Selectable for a comma-separated list of days. The time is specified as shown below.
- **Absolute** - On a specified date and time. The year may be specified or left as \* to signify any year.
- **Self-Trigger** - Start Type only. Does not start automatically. Started by the use of a special PLEG action (see later).
- **Self-ReTrigger** - Start Type only. As **Self-Trigger** but may be restarted whilst running to provide *Watchdog* functionality.

There are several options for specifying the time for **Day of Week** and **Day of Month** types:

- **At a certain time of day** - The time is specified in **Hours, Minutes and Seconds**.
- **At sunrise** - At the moment of sunrise as predicted for your location.
- **Before sunrise** - The specified **Hours, Minutes and Seconds** earlier than sunrise.
- **After sunrise** - The specified **Hours, Minutes and Seconds** later than sunrise.
- **At sunset** - At the moment of sunset as predicted for your location.
- **Before sunset** - The specified **Hours, Minutes and Seconds** earlier than sunset.
- **After sunset** - The specified **Hours, Minutes and Seconds** later than sunset.

With the exception of **Self-Trigger** and **Self-ReTrigger**, all of the types may include an optional **Random Delay** in the form hh:mm:ss. When this is specified, the start or stop times will be delayed by a random time between zero and hh:mm:ss.

The **Self-Trigger** Schedule provides the means to start the timer on a specific event. This can be useful for providing a lock-out of parts of your logic until the timer has run its course. This form of Schedule is started by the **Start Timer** action for the PLEG device on which the Schedule is defined - specifying the name of the **Self-Trigger** Schedule. The **Start Timer** action also allows the predefined **Stop Type Interval** to be optionally overridden by the time value entered in the action's **intervalTime** field. While a **Self-Trigger** timer is running, additional **Start Timer** actions for it will be ignored. In the case of a **Self-ReTrigger** timer, additional **Start Timer** actions for it will restart the timer. This can be used to implement *Watchdog* functionality: If the timer is started by some periodic event, the absence of this event for the period set by the **Stop Type Interval** will allow the timer to complete and the Schedule to become false.

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

If a Schedule does not have a **Stop Type**, it will not have an *off event*. Such a Schedule will only cause the evaluation of a Condition in which it is used when it turns on and its state will, at that point, always be true. If *At0730* is a daily Schedule for 07:30 without a **Stop Type**, the Condition:

**Morning**      *At0730*

will become true at 07:30 on the first day and stay that way forever. Any associated Action will only fire on the first day. There are two ways around this: Check the **Repeats** option for the Condition so that it will fire its Action each time *At0730* becomes true; Add a **Stop Type** to *At0730* – say an Interval of one minute – so that it will re-evaluate the Condition when it turns off and set it to false.

Create a PLEG Schedule by selecting **Inputs -> Schedules** then clicking **Create**. Select the required **Start Type** from the pull-down list and then enter or select the appropriate options and values. If an inclusive timer is required, repeat the process for **Stop Type**. Click **Accept**. To change the name of the new Schedule, click **Edit**, change the name from the default and click **Accept**.

When you create a new Schedule it will have no history. The Schedule will not be true even if the current time lies between the Start and Stop times. You can force a Schedule true or false by clicking its **Do It Now** button. Each time this is clicked, it will toggle the state of the Schedule.

When testing new PLEG logic, it often helps to temporarily set Schedule Start and Stop times to the near future. Once you are sure the logic works, you can edit the Schedules to set the actual times required.

The screenshot shows the 'Test' window of the PLEG software. The 'Inputs' tab is selected, and the 'Schedules' sub-tab is active. A table lists existing schedules with columns for Name, Type, Random On Delay, Off After Type, Random Off Delay, and Actions. The 'Create' button is visible above the table. A message at the top of the table area says 'Close dialog and press SAVE to commit changes.'

Name	Type	Random On Delay	Off After Type	Random Off Delay	Actions
I30M	Interval:30:00	None	undefined	None	<button>Edit</button> <button>Do It Now</button> <button>?</button>
HeatWD	Weekly:1,2,3,4,5	None	Weekly:1,2,3,4,5	None	<button>Edit</button> <button>Do It Now</button> <button>?</button>
Timer5M	Self Trigger	None	Interval:5:00	None	<button>Edit</button> <button>Do It Now</button> <button>?</button>
GardenLights	Weekly:1,2,3,4,5,6,7	20:00	Weekly:1,2,3,4,5,6,7	30:00	<button>Edit</button> <button>Do It Now</button> <button>?</button>

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Conditions

PLEG Conditions are where you define the logic that connects inputs and actions. Conditions have a name and an expression. The expression contains one or more terms separated by operators. A term can be a Trigger, Device Property, Schedule, constant or Condition name. Operators can be logical, arithmetic or string. There are some special forms of expression that are evaluated on the basis of *timestamps* rather than current value.

A normal Condition will *fire* when the value of its expression changes from false (0) to true (not 0). The time when the state last changed is recorded as the Condition's *timestamp*.

If a Condition's **Repeats** check-box is checked, it will *fire* every time the value of its expression changes and is not zero. Its *timestamp* will be the time when this last happened.

Conditions are evaluated from the top down. If you use the value or *timestamp* of one Condition in another, the order of the Conditions is significant. If you refer to the value or *timestamp* of a Condition that is below your expression, you will be using the results of the previous evaluation run.

PLEG will start an evaluation cycle when one of the Triggers, Device Properties or Schedules, that is used as a term in at least one Condition, changes. A Condition will only be evaluated if it includes a term that has changed since the last evaluation run.

Create a Condition by selecting the **Conditions** tab and clicking **Add Row**. Change **Condition Name** on the left as required and type your expression into **Condition Expression**. You can also use the built-in editor by clicking the **Edit** button. You can change the order of Conditions by clicking the + or – buttons. The – button will raise a Condition up the list and the + button will lower it.

The screenshot shows the 'House Heat Logic' application window with the 'Conditions' tab selected. The interface includes a toolbar with icons for help, settings, and window management. The 'Room' dropdown is set to 'no room'. The 'Conditions' tab is active, displaying a list of four conditions. Each condition row has a name, a plus/minus button, a 'Repeats' checkbox, an expression field, and 'Delete'/'Edit' buttons.

Condition Name	Repeats	Condition Expression	Buttons
SendTemp	<input type="checkbox"/>	AlarmSet and ((22:45:00; AlarmSet) or (AlarmSet; 03:59:59))	Delete, Edit
WaterOn	<input checked="" type="checkbox"/>	!WaterAM or !WaterPM or (EarlyStart and EarlyION) )	Delete, Edit
WaterOff	<input checked="" type="checkbox"/>	!HotWater or (HotWater and (!!WaterAM and !!WaterPM))	Delete, Edit
EarlyWaterOff	<input type="checkbox"/>	HotWater and EarlyStart and (EarlyStart; EarlyION; EarlyTOff)	Delete, Edit



## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Logical Expressions

In its simplest form, a Condition will contain logical terms and operators and have a logical result.

**LightsOn** *ItsNight* and *Away*

**LightsOn** will be true whenever *ItsNight* is true and *Away* is true. If either *ItsNight* or *Away* are false, **LightsOn** will be false.

The logical operators are:

- **not** (9) Negates the following term so true becomes false and vice versa.
- **!** (9) Exactly equivalent to **not**.
- **and** (3) The result will only be true if both the preceding and following terms are true.
- **or** (2) The result will be true if either of the preceding or following terms are true.

All operators have a precedence (as shown in parentheses) that dictates the order in which they will be applied. Higher numbers are applied first. So in the expression **A and not B or C**, *B* is negated then and-ed with *A* and the result is or-ed with *C*. Parentheses may be used to force the evaluation of expressions in a certain way: **A and not (B or C)** is quite different from the previous version.

Parentheses may be used liberally to make expressions more readable and to simplify expressions:

**A and B or A and C and D or A and D and E** may be written as **A and (B or (D and (C or E)))**

You may use any practical number of terms, or pairs of parentheses, in an expression.

### Numerical Comparisons

You can test the values of Device Properties or other Conditions by using numerical comparisons.

**LightsOn** *Away* and (*LightLevel* < 100)

**LightsOn** will be true whenever *Away* is true and *LightLevel* is less-than 100. If either *Away* is false or *LightLevel* is higher than 99, **LightsOn** will be false.

The numerical comparison operators are:

- **==** Is equal to.
- **!=** Is not equal to.
- **>** Is greater than.
- **>=** Is greater than, or equal to.
- **<** Is less than.
- **<=** Is less than, or equal to.

The numerical comparison operators have a precedence of (4) so they will be applied after arithmetic operators but before logical ones – unless forced by parentheses.

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Arithmetic Expressions

All PLEG terms (Triggers, Device Properties, Schedules, Conditions, expressions) have a value. For logical (boolean) terms, the value is 0 when false or 1 when true. Device Properties, Conditions and expressions may have a range of values. Provided the values are valid numbers, they may be used in arithmetic expressions.

**HeatOn**  $(TempF - 32) * 5 / 9 < TempLimit$

**HeatOn** will be true whenever *TempF*, converted from Fahrenheit to centigrade, is less-than the value of *TempLimit*. *TempLimit* could be a Device Property reflecting the setpoint of a metric thermostat or another Condition statement where the limit was calculated.

The arithmetic operators are:

- - (9) Unary negation of the following term. Positive becomes negative and vice versa.
- # (9) Return the false-to-true (on) timestamp of the following term.
- #! (9) Return the true-to-false (off) timestamp of the following term.
- % (8) Modulus. The remainder of integer division by the following term.
- \* (7) Multiply by the following term.
- / (7) Divide by the following term.
- + (6) Add the following term.
- - (6) Subtract the following term.

All operators have a precedence (as shown in parentheses) that dictates the order in which they will be applied. Higher numbers are applied first. You should note that all arithmetic operators will be applied before any comparison or logical ones. If in doubt, just add parentheses.



## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Conditional Expressions

The conditional expression returns one of two different values depending on the true/false state of the first term. It may help to consider it as an *if - then – else*. The format is:  $XX ? YY : ZZ$  which has the value  $YY$  if  $XX$  is true and  $ZZ$  if  $XX$  is false. The first term must evaluate to true or false.

The conditional expression has a precedence of (1) so will be evaluated after all other operators. It will be necessary to use parentheses in most cases when using conditional expressions with other operators.

**HeatOn**  $HouseTemp < (ItsNight ? 14 : 20)$

**HeatOn** will be true whenever *HouseTemp* is less than 14 if *ItsNight* is true or 20 if *ItsNight* is false.

Conditional expressions can be cascaded if required.

**HeatOn**  $HouseTemp < (ItsNight ? 14 : (WeekEnd ? 21 : 19))$

As with the previous example, the temperature setpoint is 14 when *ItsNight* is true but will depend on the state of *Weekend* when *ItsNight* is false. When *WeekEnd* is true, the daytime setpoint will be 21 otherwise it will be 19.

Conditional expressions can be cascaded to almost any necessary level. Use parentheses to ensure that they are evaluated in the required order and to help with readability.

The first term in a conditional expression may be any expression that has a logical (Boolean) result.

**HeatOn**  $HouseTemp < ((LightLevel < 100) ? 14 : 20)$

**HeatOn** will be true whenever *HouseTemp* is less than 14, if *LightLevel* is less than 100, or 20 if *LightLevel* is greater than 99.

### Condition Values

So far all the examples have been written to give the Conditions a Boolean (true/false) value. A Condition may take any value, though. This feature can be used to calculate values for use in other Condition expressions.

**SetPoint**  $(Home ? (ItsNight ? 14 : (WeekEnd ? 21 : 19)) : 5)$

**HeatOn**  $HouseTemp < SetPoint$

**SetPoint** is used to calculate the required temperature threshold used by **HeatOn**. This will be 5 if *Home* is false. If *Home* is true, it will be 14 when *ItsNight* is true. If *Home* is true but *ItsNight* is false, it will be 21 when *WeekEnd* is true or 19 if not.

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### String Expressions

When the value of a Device Property, Condition or expression is a sequence of characters that is not a valid number, you can only employ it by using string expressions.

**HeatOn** HouseTemp < 20 and HWRMode eq "Off"

**HeatOn** will be true whenever *HouseTemp* is lower than 20 and *HWRMode* contains the string *Off*

The string operators are:

- **..** (5) Concatenate the following string term.
- **eq** (4) Is equal to.
- **ne** (4) Is not equal to.
- **gt** (4) Is greater than.
- **ge** (4) Is greater than, or equal to.
- **lt** (4) Is less than.
- **le** (4) Is less than or equal to.

The string comparison operators have a precedence of (4) so they will be applied after any concatenation operators but before logical ones – unless forced otherwise by parentheses.

Equal-to (**eq**) and not-equal-to (**ne**) work as you would expect: Two strings are equal when they are identical. Note that case is significant in string comparisons. "Off" is not equal to "off" or "OFF".

Greater-than (**gt**) and less-than (**lt**) are a little more complex: They are compared using a lexical sort order – like a dictionary or index. Note that lower-case characters are greater-than upper-case ones.

- "Off" is greater than "Ofe"
- "Off" is greater than "Oeg"
- "Off" is less than "Ofg"
- "Off" is greater than "OfG"

String literals (constants) must be delimited with single or double quotation marks. e.g. 'Heat' or "Heat". There is no difference in the value of these two literals.

When used in logical (Boolean) expressions, a term with a string value is logically true if it has a length greater than zero. So "x" is true but "" is false. This can give unexpected results if you mistakenly treat a Device Property as if it were a Trigger. If **Status** is a Device Property with possible values of "0" or "1" and it is used in a Condition like this:

**AutoOn** Status and Motion...

**Status** will always be true because it is a string with a length greater than zero. You should instead use either of the following forms:

**AutoOn** Status eq "1" and Motion...

**AutoOn** Status == 1 and Motion...

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Working With Time

Up to now we have been concerned with the values of terms in our expressions. Sometimes, though, we care about the times at which events occurred. This is where PLEG really shines.

As has been noted in the preceding sections, whenever a Trigger, Schedule or Condition changes state, the time at which this happened is noted as a *timestamp*. Also, whenever the value of a Device Property, State Variable or Condition with **Repeats** checked changes, the *timestamp* is recorded. These *timestamps* are the foundation of PLEG's ability to work with time.

### Sequence Expressions

A sequence expression is a list of terms separated by commas or semi-colons. It may be followed by one or two time constraints. A term can be the name of a Trigger, Device Property, Schedule, State Variable or Condition. Normally the sequence expression will consider the *timestamp* of the last false/true (on) transition of the term. If the term is preceded by an exclamation mark (!), it refers to the *timestamp* of the last true-to-false (off) transition instead. A term can also be an absolute time as hh:mm:ss. It cannot be an expression. Sequence expressions are evaluated solely on the *timestamps* associated with the terms. The current value of the terms is irrelevant.

If the terms are separate by commas, the sequence expression will be true when all the terms have *timestamps* that meet the time constraints. The order of the terms and *timestamps* doesn't matter. This is called an *un-ordered* sequence expression.

If the terms are separated by semi-colons, the sequence expression will be true when the *timestamps* for the terms are in the same order as the terms and meet the time constraints. This is called an *ordered* sequence expression.

One, two or no time constraints may follow the list of terms. The constraints are written in the form:  
**< hh:mm:ss** or **> hh:mm:ss**.

**< hh:mm:ss** indicates that the time between the oldest and newest *timestamps* for the terms in the list must be less than *hh:mm:ss*.

**> hh:mm:ss** indicates that the time between the oldest and newest *timestamps* for the terms in the list must be greater than *hh:mm:ss*.

**LockDoor** DoorClosed; AlarmSet < 10

**LockDoor** will become true if *AlarmSet* becomes true less than 10 seconds after *DoorClosed* becomes true. This is not a practical example, though, because it disregards the current state of *DoorClosed* and *AlarmSet* so would be true when *AlarmSet* becomes true even if *DoorClosed* was false – but had once been true.

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

**LockDoor** DoorClosed and AlarmSet and (DoorClosed; AlarmSet < 10)

**LockDoor** will become true when *DoorClosed* and *AlarmSet* are true and happened, in that order, within 10 seconds. Parentheses are important when mixing sequence, logical and arithmetic expressions.

**LightOn** (Motion1 or Motion2 or Motion3) and (Motion1, Motion2, Motion3 < 1:00)

**LightOn** will become true if Motion1, Motion2 and Motion3 are all tripped, in any order, within one minute.

**LightOff** LightOn and (LightOn; !Timer)

**LightOff** will become true if Timer ends after LightOn became true.

There is a special predefined term that may be used in sequence expressions. **Now** is effectively an interval timer with a one-minute period. Its presence in an expression will cause the expression to be evaluated every minute. **Now** will have a *timestamp* of the current time when the evaluation takes place. Note that **Now** is not a variable containing the current time although you could use **#Now** to get its timestamp which is effectively the same thing.

**LightOff** LightOn and (LightOn; Now > 5:00)

**LightOff** will become true if *LightOn* remains true for 5 minutes.

**Lockout** Motion; Now < 2:00

**Lockout** will be true for two minutes after *Motion* becomes true.

Note that as **Now** only *fires* once per minute, it is not suitable for precise timing. The expression: **Event; Now > 1:00** could become true at any time between 1:01 and 2:00 after *Event* becomes true.

Absolute times may also be used as terms in a sequence expression.

**Away** AlarmSet and (08:00:00; AlarmSet; 21:00:00)

**Away** will become true if *AlarmSet* becomes true between 08:00:00 and 21:00:00.

A combination of absolute times and **Now** could be used as an alternative to using a **Schedule** but, as it causes an evaluation cycle once a minute, it would use considerably more CPU cycles.

**DayTime** (06:00:00; Now; 21:30:00)

**DayTime** will be true between 06:00:00 and 21:30:00.

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Multiple Trigger Expressions

Sometimes we want to recognise when a number of events have occurred in a certain period of time. PLEG provides a means to do that with a special expression.

**TriggerOrDevicePropertyOrCondition @ Count > hh:mm:ss < hh:mm:ss**

The first term is the name of a Trigger, Device Property or Condition whose events we wish to count. **Count** is the number we are testing for. **> hh:mm:ss** and **< hh:mm:ss** are time constraints for the counted events. The time constraints are applied by checking the *timestamps* of the oldest and newest of the counted events. Either or both of the time constraints may be used.

**> hh:mm:ss** indicates that we want to test if the counted events occurred in a period greater than *hh:mm:ss*.

**< hh:mm:ss** indicates that we want to test if the counted events occurred in a period less than *hh:mm:ss*.

**SoundSiren** MotionDetector @ 3 < 5:00

**SoundSiren** will become true if *MotionDetector* becomes true three times in a five minute period.

### Special Conditions

PLEG has two reserved Condition names: **ARM** and **BYPASS**. If you create conditions with these names, they will directly control PLEG's **Arm/Bypass** state without you needing to define an Action. When **ARM** becomes true, it will set PLEG to *Arm*. When **BYPASS** becomes true, it will set PLEG to *Bypass*.

### State Variables

PLEG provides a State Variable mechanism through the use of a special form of Condition naming: **StateName\$StateValue**. When such a Condition becomes true, it sets the value of **StateName** to *StateValue*. The value of the State Variable may be tested in logical expressions using **StateName eq "StateValue"**. The value of **StateName** will be initialized to the *StateValue* of the first Condition with a **StateName\$StateValue** name.

In the following Conditions, **State** will initially have the value "S1". If **Step** is a Trigger, each time it becomes true the value of **State** will cycle through "S2", "S3", "S1", "S2", etc.

**State\$S1** Step and (State eq "S3") and (State; Step)

**State\$S2** Step and (State eq "S1") and (State; Step)

**State\$S3** Step and (State eq "S2") and (State; Step)

You may define Actions for each **StateName\$StateValue** combination and they will fire when that Condition becomes true. You may also define an Action for **StateName** which will fire whenever its value is changed – behaving in the same way as a Condition with **Repeats** checked.

### Actions

Actions are how PLEG makes things happen. They are associated with Conditions and will be performed when the Conditions *fire*. A normal Condition *fires* when its value changes from false to true. A Condition with **Repeats** checked will *fire* whenever it is evaluated as true or non-zero. If an Action has been created for a Condition, it will be performed whenever the Condition *fires*.

Actions are events. They are performed when they are *fired*. If an Action turns on a lamp, that's all it does. When the Condition that *fired* the action is no longer true, there is no automatic mechanism to undo the Action. If you want the lamp turned off, you must create a Condition and associated Action to do that. You can have a single Action turn a device on or off by using the **{{yourexpression}}** feature to set *newTargetValue* according to *yourexpression* (see later).

An Action consists of one or more device actions. These operations may be **Immediate** or **Delayed**. There are many different possible device actions depending on the type of the target device. Examples include: *Turn On*, *SetCurrentSetpoint*, *SetLoadLevelTarget*.

Simple device actions may be defined using the same *point-and-click* interface used for Vera scenes. More complex operations are defined through the **ADVANCED** tab where all available action calls for a device may be selected and the associated parameters specified. PLEG provides a special mechanism for action parameters with the term: **{{yourexpression}}** where **yourexpression** is a PLEG expression that evaluates to a suitable value for the operation being performed.

Create an Action by selecting the **Actions** tab and clicking the **Edit** button for the Condition for which you want the Action. This will display the same page used for defining device actions in scenes. You can control timing of the device actions using the drop-down box at the top left. If you simply want to turn a device on or off, or set a dimmer level, you can select this by clicking on the appropriate button of the device. You can select multiple device actions on this page.

More complex device actions can be defined on the **ADVANCED** tab. Here you can select the device on the **Pick a device:** drop-down and then click **Add**. Now there is a drop-down containing all of the device actions supported by the device. When you select one, if it requires a parameter, a blank field will appear. Enter the required parameter in this field. The parameter may be a number or a string depending on the selected device action. This is where you can enter **{{yourexpression}}** if you want the parameter value to be calculated by PLEG using *yourexpression*.

PLEG provides some special device actions when it is selected as a target for an Action.

- **StartTimer** Start the *Self-Trigger* or *Self-Retrigger* **Schedule** specified by **timerName**. Optionally set the interval to **intervalTime**.
- **RunScene** Run the Vera scene specified by **SceneNameOrNumber**

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

PLEG Actions may also include Lua code. This is entered on the Action's **LUUP** tab. As with Vera scenes, if the Lua code returns false, the explicit device actions will not be performed. If the code returns true or nothing, the device actions will be performed.

You can access PLEG's Inputs and Conditions from Lua code using the following syntax where **XXXX** is the name of the item in upper-case (e.g. LightOn --> LIGHTON):

- **XXXX.state** The value of Input or Condition **xxxx**
- **XXXX.seq** The timestamp when **xxxx** last became true (on)
- **XXXX.oseq** The timestamp when **xxxx** last became false (off)
- **XXXX.seqh[n]** The nth *on* timestamp history for **xxxx** where n is between 1 and 10  
**XXXX.seqh[1]** is the same as **XXXX.seq**

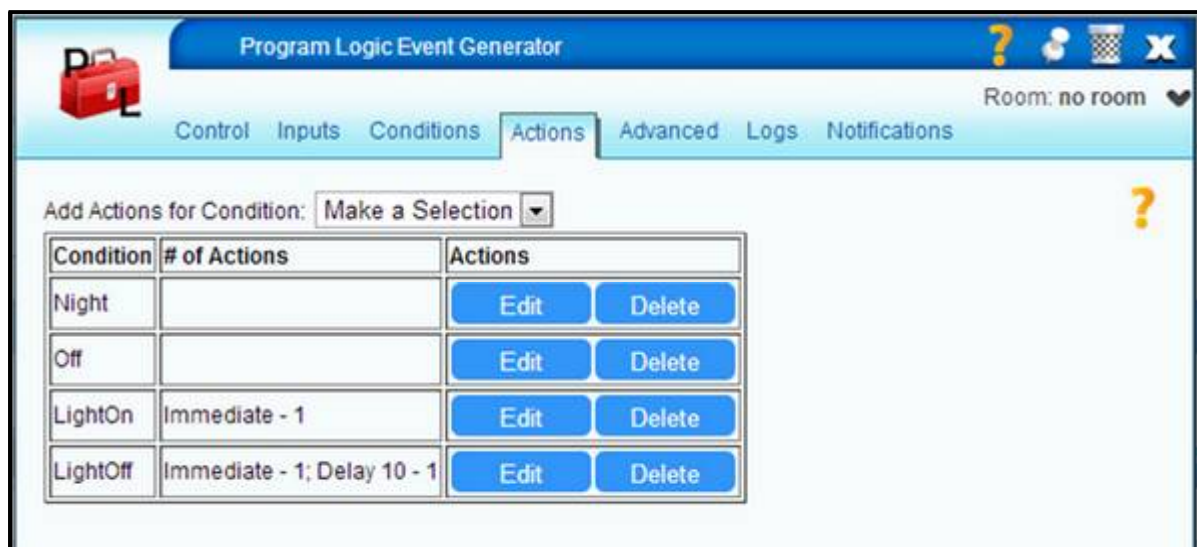
These variables should be considered read-only. Assignment to these variables may lead to unpredictable PLEG behaviour. The on timestamp history in **XXXX.seqh[n]** is not preserved through a Vera restart.

Lua code in PLEG Actions may include Vera's **luup** functions so it is possible to get or set device variables and utilise other extension functions as you might do in Vera scene Lua. Global variables set in one Action will be available to Lua in other Actions of the same PLEG device. Global variables defined in Vera scenes and **Startup Lua** are not available in PLEG's Lua context.

Once you have finished entering or editing code in the **LUUP** tab, click **Save lua**.

When you have defined all required device actions, click the **FINISHED** tab. Your new device actions should be shown in the list for the selected Condition with *Immediate* or whatever delay you chose.

Actions are only executed when PLEG is **Armed** – as shown by the **Arm** button being highlighted.





## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Working with PLEG

Whenever you have created or edited any Triggers, Device Properties, Schedules, Conditions or Actions, you must close the PLEG device window, click on Vera's **Save** button and then reload your browser page (F5 key). If you do not perform this sequence, you may get erroneous or confusing results.

PLEG provides a very useful report when you click the **Status** button on the **Control** tab. The report shows the current state of all Triggers, Device Properties, Schedules, State Variables and Conditions and the *timestamps* of when they last changed. For Device Properties and State Variables, the *timestamp* of the previous change is also shown. The *timestamps* show the date and time including milliseconds. This information is very valuable when debugging your logic – particularly when you are using sequence expressions.

Remember that PLEG will only perform an evaluation cycle when some Trigger, Device Property or Schedule has changed and is used in one of the Condition expressions. An evaluation cycle is also triggered every minute if the special term **Now** has been used in a sequence expression.

Don't forget to **Arm** the PLEG before you test your logic. If PLEG is set to **Bypass**, none of the Actions will be executed. The special Condition **ARM** can be used to Arm a Bypassed PLEG as it does not require an Action.

PLEG's **Log** button may be used to display its own log entries. It will also allow you to select verbose **Debug** logging to either Vera's main *LuaUPnP.log* or a *Standalone* file. When you change **Debug**, you must click Vera's **Save/Reload** button for the change to take effect.



## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Licensing

Starting from April 1<sup>st</sup> 2014, PLEG versions greater than 4.5 will only provide the unrestricted features if a valid license has been purchased and registered. The license is a one-time purchase for use on a single Vera serial number and will remain valid for subsequent versions of the firmware. Each license allows the use of up to four devices which may be a mixture of PLEG and PLTS. Multiple licenses may be purchased to expand the number of permitted devices in increments of four.

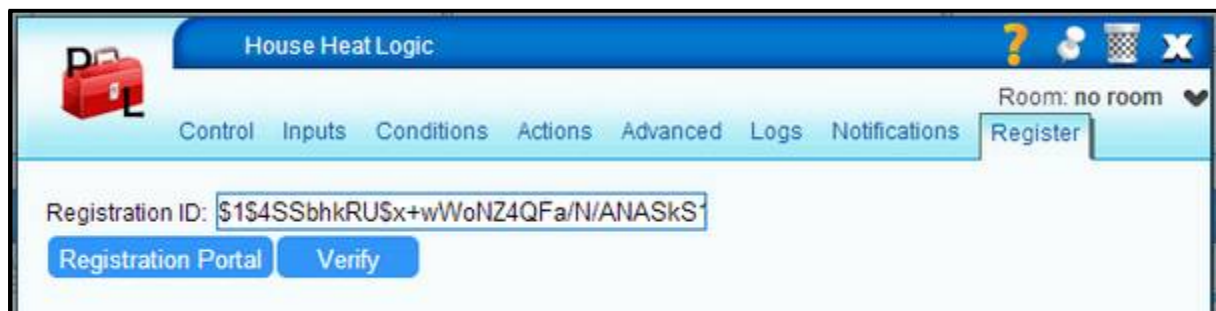
To allow for proper evaluation of PLEG, the full feature set may be used without the purchase of a license for thirty days after installation. If PLEG was installed on your system earlier than March 2<sup>nd</sup> 2014, there will be no additional evaluation period after April 1<sup>st</sup> 2014.

If a license has not been registered, and PLEG has been installed for more than thirty days, the following restrictions will be applied from April 1<sup>st</sup> 2014:

- A maximum of three PLEG or PLTS devices may be used
- Each PLEG will be limited to five Triggers and five Conditions
- Devices that exceed these restrictions will be disabled

If a system includes more PLEG or PLTS devices than permitted by the registered license, only the number entitled by the license will run. Messages on the Vera UI Dashboard will show if any devices have been disabled.

Licenses may be purchased by clicking the **Registration Portal** button on the **Register** tab that is included on PLEG V6.0 and later. When the license code is received by email, enter it into the **Registration ID** field and click **Verify**. The registration status is shown on PLEG's **Status** report. The **Registration ID** is unique to the Vera serial number for which it was purchased. It has no value to anyone else so does not need to be obscured when posting Status reports.



## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Example – Simple Motion-Triggered Light

Turn on a light when a motion-detector is tripped at night. Turn it off if there has been no motion for 10 minutes. Ignore motion for 30 seconds after light is turned off. The light may be locked on by rapid on/off/on of the light switch (requires a light switch with instant or very rapid status reporting). The motion sensor should be configured with a short on-time (less than the interval time). None of the Conditions require **Repeats** to be checked.

#### Triggers

<b>LightOn</b>	Light is turned on
<b>Motion</b>	Motion Sensor is tripped
<b>ItsNight</b>	DayTime indicates night

#### Schedules

<b>Timer</b>	On: Self-ReTrigger Off: Interval 10:00
--------------	----------------------------------------

#### Conditions

<b>AutoOn</b>	!LightOn and ItsNight and Motion and (!LightOn; Motion > 30)
<b>KeepOn</b>	(LightOn and Motion and (LightOn; Motion)) or AutoOn
<b>AutoOff</b>	LightOn and !Timer and (LightOn; !Timer) and (!LightOn; LightOn > 10)

#### Actions

<b>AutoOn</b>	Turn Light on
<b>KeepOn</b>	PLEG StartTimer timerName=Timer
<b>AutoOff</b>	Turn Light off

### Example – Light Controlled by Door Switches

Turn on a light if any of three doors are opened. Turn off the light when all doors are closed provided one or more was open for at least 20 seconds. Turn off the light if still on after one hour. None of the Conditions require **Repeats** to be checked.

#### Triggers

<b>LightOn</b>	Light is turned on
<b>Door1</b>	Door1 sensor is tripped
<b>Door2</b>	Door2 sensor is tripped
<b>Door3</b>	Door3 sensor is tripped

#### Conditions

<b>AutoOn</b>	Not LightOn and (Door1 or Door2 or Door3)
<b>AllClosed</b>	Not Door1 and Not Door2 and Not Door3
<b>AutoOff</b>	LightOn and ( (LightOn; Now > 1:00:00) or (AllClosed and (AutoOn; AllClosed > 20)) )

#### Actions

<b>AutoOn</b>	Turn light on
<b>AutoOff</b>	Turn light off

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Example – Simple Heater Timer

Turn Heater on and off according to schedules for weekdays and weekends provided Home VirtualSwitch is on. Turn Heater off if Home is turned off after Heater is turned on by any means. None of the Conditions require **Repeats** to be checked.

#### Triggers

**Home** Home VirtualSwitch is turned on  
**HeaterOn** Heater is turned on

#### Schedules

**WeekdayAM** Start Type: Day of Week, Check Days Monday to Friday, Time: 06:30,  
Stop Type: Day of Week, Check Days Monday to Friday, Time: 08:00  
**WeekdayPM** Start Type: Day of Week, Check Days Monday to Friday, Time: 20:30,  
Stop Type: Day of Week, Check Days Monday to Friday, Time: 22:00  
**WeekdendAM** Start Type: Day of Week, Check Days Saturday & Sunday, Time: 07:30,  
Stop Type: Day of Week, Check Days Saturday & Sunday, Time: 09:30  
**WeekendPM** Start Type: Day of Week, Check Days Saturday & Sunday, Time: 19:30,  
Stop Type: Day of Week, Check Days Saturday & Sunday, Time: 23:00

#### Conditions

**Away** Not Home  
**AutoOn** Home and Not HeaterOn and (WeekdayAM or WeekdayPM or WeekendAM or WeekendPM)  
**AutoOff** Not AutoOn or (HeaterOn and Away and (HeaterOn; Away))

#### Actions

**AutoOn** Turn Heater on  
**AutoOff** Turn Heater off

### Example – Set Dimmer to Level Saved in VariableContainer

#### Triggers

**Door** Door sensor is tripped  
**LightOn** Light is turned on

#### Device Properties

**DimLevel** VariableContainer Variable1

#### Conditions

**AutoOn** Door and Not LightOn  
**AutoOff** LightOn and (LightOn; Now > 5:00)

#### Actions

**AutoOn** Light SetLoadLevelTarget newLoadlevelTarget={DimLevel}  
**AutoOff** Turn Light off

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Example – Send Setpoint to Thermostatic Radiator Valve with Automatic Retry

Calculate temperature setpoint according to day/night and state of Home VirtualSwitch. Send required setpoint to TRV whenever it changes. Resend setpoint every five minutes until successfully set using **Self-Trigger** timer. None of the Conditions require **Repeats** to be checked.

#### Triggers

**Home** Home VirtualSwitch is turned on  
**ItsNight** Day or Night indicates night

#### Device Properties

**SetPoint** TRV CurrentSetpoint

#### Schedules

**Timer5M** Start Type: Self-Trigger, Stop Type: Interval, 5 Minutes

#### Conditions

**Target** ItsNight ? 14 : (Home ? 20 : 16)  
**SendSP** (Target != LastTarget) or (!Timer5M and (!Timer5M; Now < 1:05) and (SetPoint != Target))  
**LastTarget** Target

#### Actions

**SendSP** TRV SetCurrentSetpoint NewCurrentSetpoint={{Target}}  
PLEG StartTimer timerName=Timer5M

**Target** will be set to the required temperature setpoint according to the state of **ItsNight** and **Home**: If **ItsNight** is true, **Target** will be 14 otherwise it will be 20 if **Home** is true or 16 if not.

**SendSP** will become true if **Target** has changed since the last evaluation cycle or if the five minute timer **Timer5M** has just ended and the TRV setpoint does not equal **Target**. When **SendSP** becomes true, the new setpoint value from **Target** is sent to the TRV and **Timer5M** is started.

**LastTarget** is set to the current value of **Target** to allow a subsequent change to be detected.

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Example – A Simple Alarm using a State Variable

The State Variable **Alarm** is set to the states “Idle”, “Armed”, “Error”, “Tripped” or “Alert” by the state-change Conditions. Starting from “Idle”, the state will change to “Armed” if the VirtualSwitch **Arm** is turned on and none of the monitored doors are open. If any door is open when **Arm** is turned on, the state will change to “Error”. In the “Armed” state, an opened door will change the state to “Tripped” which will start a 30 second timer. If the timer ends before the **Arm** switch is turned off, the state will change to “Alert” and the associated action will send an SMS via Twilio. If **Arm** is turned off, the state will return to “Idle”. State changes will trigger a Sonos Say action to speak the value of **Alarm**. None of the Conditions require **Repeats** to be checked.

#### Triggers

<b>Arm</b>	Alarm VirtualSwitch is turned on
<b>Door1</b>	Door1 sensor is Tripped
<b>Door2</b>	Door2 sensor is Tripped
<b>Door3</b>	Door3 sensor is Tripped

#### Schedules

<b>Timer30S</b>	Start Type: Self-Trigger, Stop Type: Interval, 30 Seconds
-----------------	-----------------------------------------------------------

#### Conditions

<b>AnyDoor</b>	Door1 or Door2 or Door3
<b>Alarm\$Idle</b>	Not Arm
<b>Alarm\$Armed</b>	Alarm eq “Idle” and Arm and Not AnyDoor
<b>Alarm\$error</b>	Alarm eq “Idle” and Arm and AnyDoor
<b>Alarm\$Tripped</b>	Alarm eq “Armed” and AnyDoor
<b>Alarm\$Alert</b>	Alarm eq “Tripped” and (Alarm\$Tripped; !Timer30S)

#### Actions

<b>Alarm\$Tripped</b>	PLEG StartTimer timerName=Timer30S
<b>Alarm\$Alert</b>	Twilio SendSMSMessage Message= “Call Police...”
<b>Alarm</b>	Sonos Say Text={{‘Alarm state is ‘ .. Alarm}}

## PLEG Basics – An Introduction to the Program Logic Event Generator V6.2

### Example – A Bathroom Fan Timer

This example uses the State Variable **Fan** and the Self-ReTrigger Schedule **Timer** to control a bathroom fan according to how long the bathroom light is turned on. A brief visit to the bathroom does not cause the fan to start. A visit over three minutes but less than ten will cause the fan to be run for ten minutes. A visit of over ten minutes will run the fan for thirty minutes. The fan can be run for ten minutes by switching the light on/off/on quickly. If the fan is turned on manually it will be turned off after twenty minutes. None of the Conditions require **Repeats** to be checked.

When the light is first turned on, **Fan** is set to '*Idle*' and **Timer** is started for three minutes. If **Timer** ends and the light is still on, the bathroom fan is turned on, **Fan** changes to '*Short*' and **Timer** is started for ten minutes. **Fan** will also be set to '*Short*', the fan turned on and **Timer** started for ten minutes if the bathroom light is turned off and back on within ten seconds while the fan is turned off.

When **Fan** is '*Short*' and **Timer** ends, if the light is still on, **Fan** changes to '*Long*' and **Timer** is started for twenty minutes otherwise **Fan** changes to '*Off*' and the fan is turned off.

**Fan** will also be set to '*Long*' and **Timer** started for twenty minutes if the bathroom fan is turned on manually.

When **Fan** is '*Long*' and **Timer** ends, **Fan** changes to '*Off*' and the fan is turned off regardless of whether the light is on or off.

#### Triggers

<b>LightOn</b>	Bathroom Light is turned on
<b>FanOn</b>	Bathroom Fan is turned on

#### Schedules

<b>Timer</b>	Start Type: Self-ReTrigger, Stop Type: Interval, 3 Minutes
--------------	------------------------------------------------------------

#### Conditions

<b>Fan\$Off</b>	FanOn and !Timer and ((Fan eq ' <i>Short</i> ' and !LightOn and (Fan\$Short; !Timer)) or (Fan eq ' <i>Long</i> ' and (Fan\$Long; !Timer)))
<b>Fan\$Idle</b>	!FanOn and LightOn and (!FanOn; LightOn)
<b>Fan\$Short</b>	(Fan\$Idle and !Timer and (Fan\$Idle; !Timer)) or (!FanOn and (!LightOn; LightOn < 10))
<b>Fan\$Long</b>	(Fan eq ' <i>Short</i> ' and !Timer and LightOn and (Fan\$Short; !Timer)) or (FanOn and (Fan eq ' <i>Off</i> ' or Fan eq ' <i>Idle</i> ') and (Fan\$Off; FanOn))

#### Actions

<b>Fan\$Off</b>	Bathroom Fan off
<b>Fan\$Idle</b>	PLEG StartTimer timerName=Timer intervalTime=3:00
<b>Fan\$Short</b>	Bathroom Fan on PLEG StartTimer timerName=Timer intervalTime=10:00
<b>Fan\$Long</b>	PLEG StartTimer timerName=Timer intervalTime=20:00