

Plugwise Template Engine

Title	Plugwise Template Engine
Version	2.0
Date	2009-05-12
Product	Source/PTE
Author	TVR
Notes	This is an experimental feature and is not considered as required functionality. There will not be any support from the Plugwise helpdesk.
Bugs	Please report your remarks and bugs to helpdesk@plugwise.com
Changes	0.94: While loop statement added 0.95: PlugwiseServer.exe added 0.96: File object added, additional properties for System object. 2.00: Big performance improvements 2.01: Added switching and usage members to Group en Room 2.02: Added Type and TypeText to Room

Introduction

The Plugwise Source application has a built in single threaded lightweight web server with a simple object oriented template engine. This web server can be used to expose information on the Plugwise system and switch appliances remotely by means of HTML pages or XML feeds.

Installation

The web server is part of the Source application and does not require a separate installation. It is automatically started if it is enabled in the Settings window, the given port number is available and the specified 'www' folder exists.

These settings can be bypassed by specifying an ini file in the command line with

```
/httpdini="path to ini"
```

Example:

```
; Example ini file
[server]
; port number to listen on
port=8080
; folder that contains the files to serve.
; it may be relative to the application startup folder
root=www
; user name for authentication
; if left blank, no authentication is required
user=admin
; MD5 hash of the password for authentication.
; the default is 'admin'
password=21232F297A57A5A743894A0E4A801FC3

[settings]
; any parameter specified here is accessible within the scripts
; via the System.Settings array.
```

```
CompanyName=ACME inc.  
CompanyColors=#ff00ff,#800080,#00FF00,#008000
```

There is a dedicated application: PlugwiseServer.exe, which only runs the web server and does not have the user interface of the Source. PlugwiseServer uses the same command line parameters as Source.

The Basics

Any file requested by a client (i.e. web server) that has one of the extensions '.css', '.html', '.txt' or '.xml' is parsed by the template engine and any text enclosed by '<%>' and '%>' tags is interpreted as statements. All characters outside these tags and files with other extensions are literally passed through.

```
<html><body>  
<%  
    $mytext="Hello world"  
%>  
<h1><%= $mytext %></h1>  
</body></html>
```

You can enclose multiple statements with the tags as long as they are separated by a line break (end of line) or a semicolon ';':

```
<html><body>  
<%  
    $mytext="Hello world" // everything on this line behind the '///' is ignored.  
    Echo "<h1>", $mytext, "</h1>"; $a=5; Echo $a  
%>  
</body></html>
```

The default page for any folder is 'index.html'.

Variables

Variables are dynamic and weak typed, what means that you do not need to declare them and that they can change from one type to another depending on the last assignment except for array elements, their type is determined at creation and will not change.

All variables are treated as objects although there is distinction between the value types 'float', 'string' and 'bool' and reference types like 'array' or 'Appliance'. Value types have their value copied from one variable to another, while reference types get only a reference (pointer) to the object.

```
<html><body>  
<%  
    $value1=1;  
    $value2=$value1;  
%>  
Value1 = <%= $value1 %><br>  
Value2 = <%= $value2 %>  
<hr>  
<%  
    ++$value2;  
%>  
Value1 = <%= $value1 %><br>  
Value2 = <%= $value2 %>  
<hr>  
<%  
    $ref1={ 'One', 'Two' };  
    $ref2=$ref1;  
%>  
Ref1[1] = <%= $ref1[1] %><br>  
Ref2[1] = <%= $ref2[1] %>  
<hr>
```

```
<%  
$ref1[1]='Changed';  
%>  
Ref1[1] = <%= $ref1[1] %> <br>  
Ref2[1] = <%= $ref2[1] %>  
</body></html>
```

The output will look like:

```
Value1 = 1  
Value2 = 1  


---

Value1 = 1  
Value2 = 2  


---

Ref1[1] = Two  
Ref2[1] = Two  


---

Ref1[1] = Changed  
Ref2[1] = Changed
```

When operators are used on 2 values of different types, the second value is converted to the same type as the first value.

Array

An array is an indexed list of values (elements). Arrays can be associative what means that an element can not only be addressed by its index (number) but also by its key (string), if it has one. Single elements can be accessed by specifying the index or key surrounded by square brackets, '[' and ']' following the array value. The zero based index is created automatically and may change every time the array is modified. Keys are case insensitive, are assigned by statements and are valid until the associated array element is removed from the array. Elements in the same array can be of different types.

An array is assigned by specifying the elements between curly brackets, separated by a comma:

```
$b={ 'One'=>'1', 2, 3, 'Four'=>'4' }
```

Or a single element:

```
$b['Five']=5
```

Operator	Description	Example	Result
+ +=	Add one or more elements.	<pre>\$a={1}+{2,3} \$a+= {4,5}</pre>	<pre>{0=>1,1=>2,2=>3} {0=>1,1=>2,2=>3,3=>4,4=>5}</pre>
- -=	Remove one or more elements. If a key is given, the value is ignored.	<pre>\$c=\$a-{2,5} \$b-={'Two'=>"Don't care"}</pre>	<pre>{0=>1,1=>3,2=>4} {'One'=>'1'}</pre>
==	Is Equal to. Two array are equal if they have the same number of elements and all values in the first array exists in the second array and vice-versa. The indices and/or keys are ignored.	<pre>\$a={'1'} \$a={3,1,2} \$b={1,2,3} \$a==\$b</pre>	True True
!=	Is not equal to, reverse of '=='		

Member	Description	Example	Result
ClassName	The class name of the object		
ContainsKey(key)	True if the array contains an element with key <i>key</i>		
ContainsValue(value)	True if the array contains		

	an element with value <i>value</i>		
Count	Number of elements	<code>\$a={"abc",5,"xy"};</code> <code>\$a.Count</code>	3
First	First element	<code>\$a.First</code>	"abc"
GetUnique()	Returns a copy of the array minus the duplicate elements		
Join(<i>sep</i>)	Concatenate all the values to one string using <i>sep</i> as separator.	<code>\$a.Join(";")</code>	"abc;5;xy"
Keys	Array of all keys. For elements without a key, the index is returned.	<code>\$b={"One"=>'1','Two'=>'2',7}</code> <code>\$b.Keys</code>	{ 'One', 'Two', 2 }
Last	Last element	<code>\$a.Last</code>	"xy"
Values	Array of all values.	<code>\$b.Values</code>	{ '1', '2', 7 }

Bool

Bool is short for Boolean and can have only one of two values: it is either 'true' or 'false'.

Operator	Description	Example	Result
<code>==</code>	Is equal too	<code>\$a==True</code>	False
<code>!=</code>	Is not equal to	<code>\$a!=False</code>	
<code>!</code>	Logical NOT		
<code>&&</code>	Logical AND		
<code>//</code>	Logical OR		
<code>(bool)?expr1:expr2</code>	If <i>bool</i> equals True the result of the whole expression will be the result of <i>expr1</i> . Otherwise it will be the result of <i>expr2</i> . Note: Because the engine lacks operator precedence you must enclose the <i>bool</i> expression with round brackets.	<code>\$f=4</code> <code>\$s=(\$f==4)? "Yes" : "No"</code>	"Yes"

Member	Description	Example	Result
ClassName	The class name of the object		

DateTime

A DateTime is a object which contains a specific date and time and is used for date and time calculations. When converted to a float, the resulting float contains the number of seconds since the Gregorian date 0001-01-01 00:00:00. When converted to a string the string has the sortable format "YYYY-MM-DD hh:mm:ss".

A DateTime is assigned to a variable using a constructor

```
$d=DateTime([expression])
```

Where *expression* is a float representing the number of seconds since the Gregorian date 0001-01-01 00:00:00 or a string containing a date in the sortable format "YYYY-MM-DD hh:mm:ss". If *expression* is omitted, DateTime() returns the current date and time.

Operator	Description	Example	Result
<code>+</code> <code>+=</code>	Add a date or a number of seconds Note: Since the first date is '0001-01-01', you must add 1 to the number of years, months or days you want to add when using the string format.	<code>\$d=DateTime();</code> <code>\$d2=\$d+DateTime("0010-01-01");</code> <code>\$d2+=3600;</code>	"2008-06-11 16:28:38" "2017-06-11 16:28:38" "2017-06-11 17:28:38"
<code>-</code> <code>-=</code>	Subtract a date or a number of seconds. See '+'. Note: Since the first date is '0001-01-01', you must subtract 1 from the number of years, months or days you want to subtract when using the string format.	<code>\$d-=DateTime("12:00:00");</code>	"2008-06-11 04:28:38"

- Experimental and Preliminary -

==	Is Equal to.	\$d.Date==DateTime("2008-06-11")	True
!=	Is not equal to, reverse of '=='	\$d!="2008-06-11"	True

Member	Description	Example	Result
ClassName	The class name of the object		
Date	The date part	\$d=DateTime(); \$dd=\$d.Date;	"2008-06-11 16:28:38" "2008-06-11 00:00:00"
Day	The day of the month	\$dy=\$d.Day;	11
Hour	The hour of the day	\$h=\$d.Hour;	16
Minute	The minute of the hour	\$mi=\$d.Minute;	28
Month	The month of the year	\$mo=\$d.Month;	6
Second	The second of the minute	\$s=\$d.Second;	38
Time	The time part	\$t=\$d.Time;	"0001-01-01 16:28:38"
TotalSeconds	The seconds passed since 0001-01-01 00:00:00	\$s=\$d.TotalSeconds;	63348798518
UTC	Convert to UTC Time	\$dd=\$d.UTC	"2008-06-11 14:28:38"
WeekDay	Day of the week based on Sunday as day '0'	\$wd=\$d.WeekDay	3
Year	Year of the date	\$y=\$d.Year	2008

Float

A float represents a floating point numerical value and is the only numerical type the engine supports. All numerical values are converted to floats. When an integer is required, the float is rounded to the nearest integer.

Operator	Description	Example	Result
+ +=	Add	\$f=1+0.5 \$f+=1 \$f=5+"4"+3 \$f="5"+4	1.5 2.5 48 (! 5 + "43") "54"
++	Increment by 1	++\$f	11
- -=	Subtract	\$f=20-2 \$f-=10	18 8
--	Decrement by 1	--\$f	7
==	Is equal too	1.5==2	False
!=	Is not equal to	1.5!=2	true
>	Greater than (case insensitive)	10>4	true
<	Less than (case insensitive)	10<4	false
>=	Greater than or equal to	2>=2	true
<=	Less than or equal to	10<=4	false
* *=	Multiply	\$f=5*4 \$f*=-3	20 -60
/ /=	Divide	\$f=20/5 \$f/=2	4 2
% %/	Remainder (modulus)	\$f=20%7 \$f%=4	6 2
& &+	Binary AND	\$f=63&36 \$f&=8	36 0
 =	Binary OR	\$f=13 7 \$f =16	15 31
^ ^=	Binary exclusive OR (XOR)	\$f=15^7 \$f^=15	8 7

Member	Description	Example	Result
ClassName	The class name of the object		

String

A string is the most common variable type since it normally contains readable text. Strings must be enclosed by single "" or double "" quotations marks. Comparison between strings are case insensitive. When using double quotes special characters can be escaped using the

back slash '\': \f (form feed), \n (new line), \r (carriage return), \t (tab), \\ (backslash), \" (double quote). When using single quotes, only the single quote character can be escaped.

Operator	Description	Example	Result
+ +=	Concatenate 2 strings	<code>\$s="a"+"b"</code> <code>\$s="4"+5</code> <code>\$s=4+"5"</code> <code>\$s+="a"</code>	"ab" "45" 9 "45a"
- -=	Remove all occurrences of the second string from the first.	<code>\$s="Hello World"- "l"</code> <code>\$s-= "o"</code>	"Heo Word" "He Wrld"
==	Is equal too	<code>"ab"=="ab"</code>	False
!=	Is not equal to	<code>"ab"!="ab"</code>	True
>	Greater than	<code>"ac">"ab"</code>	True
<	Less than	<code>"ac"<"ab"</code>	False
>=	Greater than or equal to	<code>"ab">="ab"</code>	True
<=	Less than or equal to	<code>"ac"<="ab"</code>	False
* *=	Concatenate a string multiple times	<code>\$s="-"*4</code> <code>\$s*=2</code>	"----" "-----"
[<i>index</i>]	The character at position <i>index</i>	<code>\$s="abcdef"</code> <code>\$s[3]</code>	"d"

Member	Description	Example	Result
ClassName	The class name of the object		
IndexOf(<i>string</i>)	The zero based start position of the first occurrence of <i>string</i>	<code>\$s="Hello world";</code> <code>\$s.IndexOf("l");</code>	2
LastIndexOf(<i>string</i>)	The start position of the last occurrence of <i>string</i>	<code>\$s.LastIndexOf("l");</code>	9
Length	The length	<code>\$s.Length</code>	11
Lower	The lower case version	<code>\$s.Lower</code>	"hello world"
MD5	The MD5 hash of the string		
Replace(<i>string1</i> , <i>string2</i>)	Replaces each occurrence of <i>string1</i> with <i>string2</i>	<code>\$s.Replace("o", "0")</code>	"Hell10 w0rld"
Split(<i>string</i> [, <i>int</i>])	Split a string on separator <i>string</i> to an optional maximum of <i>int</i>	<code>\$s.Split("l");</code> <code>\$s.Split("l", 2);</code>	{0=>'He', 1=>' ', 2=>'o wor', 3=>'d'} {0=>'He', 1=>'lo world'}
Substring(<i>int1</i> [, <i>int2</i>])	The string part starting from <i>int1</i> optionally with a maximum length of <i>int2</i>	<code>\$s.Split(6);</code> <code>\$s.Split(6, 2);</code>	"world" "wo"
Trim()	Remove white spaces from beginning and end of string	<code>" Hello\n".Trim()</code>	"Hello"
Upper	The upper case version	<code>\$s.Upper</code>	"HELLO WORLD"
UrlDecode()	Decodes the URL encoded string		
UrlEncode	URL encodes the string		

Keywords

=

<%= *expression* %>

The equals character '=' is not really a keyword but an assignment operator. However, if it immediately follows the opening tag '<%', the result of *expression* is converted to a string and passed through to client.

Example	Output
<code><%= "Hello world" %>
</code> <code><% \$a=5 %></code> <code><%= \$a %>
</code>	Hello world 5

Block, /Block

```
<% Block string %>
...
<% /Block %>
```

Defines a script part (block) with name *string* to be used (executed) later with Write. The part can contain anything except another block definition. `Block` and `/Block` must be enclosed with their own tags.

Blocks are stored in the array `System.Blocks`

Example	Output
<pre><% Block "number" %> The number is <%= \$a %>
 <% /Block %> <% \$a=5; Write System.Blocks["number"]; \$a=3; Write System.Blocks["number"]; %></pre>	<pre>The number is 5 The number is 3</pre>

Echo

```
Echo string [, string] ...
```

Writes to output. The result of expression *string* is written to output. Multiple expressions can be written by separating them with a comma. This is faster than using the '+' operator and does not cause unintentional type conversions

Example	Output
<pre><% Echo "Hello world!" %></pre>	<pre>Hello world!</pre>

Exit

```
Exit [string]
```

Terminates the script immediately and optionally outputs the message *string*.

Example	Output
<pre><% Echo "Hello world!" Exit; Echo "This is not shown" %></pre>	<pre>Hello world</pre>

ForEach, [Continue], [Break], /ForEach

```
ForEach array
    Loop
/ForEach
```

`ForEach` is a loop statement. For each element in the array resulting from expression *array*, *Loop* is executed. Within *Loop* the execution of the current loop can be stopped by `Break` and `Continue`; the first will exit the `ForEach` statement and continue the script after `/ForEach`, while the latter will restart the loop with the next element, if there

is one, from the array. *Break* and *Continue* are optional and can occur more than once within *Loop*.

Within *Loop* the index, key and value of the current element are copied to the variables *\$_Index*, *\$_Key*, resp. *\$_Value*.

ForEach constructs can be nested.

Example	Output
<pre><% \$a={ '1'=>'One' , '2'=>'Two' , '3'=>'Three' , '4'=>'Four' } ForEach \$a if \$_Index==1 continue /If %> \$a[<%= \$_Index%>] = {<%= \$_Key%>=><%= \$_Value%>}
 <% if \$_Value=='Three' break; /If /ForEach %></pre>	<pre>\$a[0] = {1=>One} \$a[2] = {3=>Three}</pre>

Format

Format name=format

Format gives a powerful method for outputting certain info in a consistent layout. Each time a value is written to output with *<%= value %>* and with *Echo* , it is formatted using the specified *format*. For formatting the rules of the .Net method *String.Format ()* are used.

Example	Output
<pre><% \$a={ 'a' , 'c' , 'd' } \$f=1.574 Format "Float.f" As "{0:0.0}" Format "Float" As "{0:0.00}" // All other floats! Format "Array.Count" As "{0}" Echo "\$a.Count=", \$a.Count, "
" Echo "\$f=", \$f, "
" %> \$a.Count=<%= \$a.Count%>
 \$f=<%= \$f%>
</pre>	<pre>\$a.Count=3 \$f=1.574 \$a.Count='3' \$f=1,6</pre>

If, [Elseif], [Else], /If

```
If bool1
  Part1
[Elseif bool2
  Part2
...]
[Else
  Partx]
/If
```

'If' is a conditional statement. If expression *bool1* results in True, then *Part1* is executed, the rest is skipped up till the */If*. If *bool1* results in False then *Part2* is executed only if *bool2* results in True, the rest is skipped up till the */If*. The *Elseif*

clause can be repeated as many times as you want. If neither the `If`-expression and none of the `ElseIf` expressions were `True`, the `Else` clause *Partx* is executed. The `ElseIf` and `Else` clauses are optional.

`If`'s can be nested.

Example	Output
<pre><% \$a=3;\$b=1 Echo "\$a is " if \$a==2 Echo "Two" elseif \$a==3 Echo "Three" if \$b==1 Echo " \$b is One" /if else Echo "Some other value" /if %></pre>	<pre>\$a is Three \$b is One</pre>

Include

```
include path
```

`Include` includes the file *path* into the current page. The code in the include file is processed as though it is part of the current page. This is especially useful script parts like block and format definitions are reused in several pages.

While, [Continue], [Break], /While

```
While bool
    Loop
/While
```

`While` is like `ForEach` a loop statement, but instead of looping through a predetermined number of array elements it loops until the given Boolean expression *bool*, results in `False`. Within *Loop* the execution of the current loop can be stopped by `Break` and `Continue`; the first will exit the `While` statement and continue the script after `/While`, while the latter will restart the loop at the point of evaluating expression *bool*. `Break` and `Continue` are optional and can occur more than once within *Loop*.

`While` constructs can be nested.

Example	Output
<pre><% \$a={'1'=>'One','2'=>'Two','3'=>'Three','4'=>'Four'} \$ix=\$a.Count While \$ix>0 --\$ix if \$ix==1 continue /If %> \$a[<%= \$ix %>] = {<%= \$a[\$ix] %>}
 <% If \$a[\$ix]=='Three' break; /If /While %></pre>	<pre>\$a[0] = {One} \$a[2] = {Three}</pre>

With, /With

```
With context
...
/With
```

Sets the current context to the result of the expression *context*. The context is the value to witch undetermined members are associated. This is especially useful when working with blocks. You can use the same block for objects that have the same member names as used within the block.

Example	Output
<pre><% \$a={ 'd' } \$b={ 'a', 'c', 'd' } With \$a Echo .Count, "
" /With With \$b Echo .Count, "
" /With %></pre>	<pre>1 3</pre>

Write

```
Write string [, string] ...
```

Writes to output. The difference with Echo, is that with Write the result of expression *string* is parsed by the engine as if it was a template file. This is why blocks should be written to output with Write and not with Echo.

Example	Output
<pre><% Block "number" %> The number is <%= \$a %>
 <% /Block %> <% \$a=5; Write System.Blocks["number"]; \$a=3; Echo System.Blocks["number"]; %></pre>	<pre>The number is 5 The number is</pre>

Engine objects

File

Static object for common file functions.

Method	Description	Example	Result
CreatePath(<i>path</i>)	Creates all the directories in <i>path</i> . Returns True if successful, False otherwise.		
Date(<i>path</i>)	Last modification date of a file		
Delete(<i>path</i>)	Deletes a file or directory. Returns True if successful, False otherwise. Note: If a directory is deleted all child directories and files are delete too.		
Exists(<i>path</i>)	Returns True if the file exists, False otherwise.		
IsDirectory(<i>path</i>)	True if an existing directory		

<code>IsFile(path)</code>	True if an existing file		
<code>Move(path, destination)</code>	Move or rename a file or directory. <i>destination</i> must be the full path to the new name. If <i>destination</i> exists, it is deleted first. Returns True if successful, False otherwise.		
<code>Read(path)</code>	Reads the contents of a text file into an array; one line per element. The CR and/or LF characters are trimmed.		
<code>Size(path)</code>	The length in bytes of a file		
<code>Write(path, array)</code>	Writes an array to a file. One line for each element. CR and LF characters are added.		

System

System is the main object of the template engine.

Method	Description	Example	Result
<code>Blocks</code>	Array of all the defined blocks	See Write	
<code>DataFolder</code>	Local path to the application data folder	<code>System.DataFolder</code>	C:\Documents and Settings\me\Application Data
<code>Date</code>	String with current local date	<code>System.Date</code>	16-06-2008
<code>EnvVars</code>	Array of the systems environment variables		
<code>Path</code>	Local path to the server root folder	<code>System.Path</code>	C:\Program Files\Plugwise\Plugwise Source\ www
<code>Settings</code>	Array with all the name-value pairs as specified in the ini file under the [Settings] category.		
<code>TempFolder</code>	Path to the temporary files folder	<code>System.TempFolder</code>	C:\Documents and Settings\me\Local Settings\Temp
<code>Time</code>	String with current local time	<code>System.Time</code>	21:37:33
<code>Version</code>	Version string of the engine	<code>System.Version</code>	0.9

Math

Math is a static object is has no value, only members and is used for mathematical calculations.

Method	Description	Example	Result
<code>Abs(float)</code>	The absolute value of <i>float</i>	<code>\$d=Math.Abs(-5);</code>	5
<code>Ceil(float)</code>	The smallest integer greater than or equal to <i>float</i>	<code>Math.Ceil(-5.3)</code> <code>Math.Ceil(5.3)</code>	-5 6
<code>E</code>	The natural logarithmic base e		
<code>Floor(float)</code>	The largest integer less than or equal to <i>float</i>	<code>Math.Ceil(-5.3)</code> <code>Math.Ceil(5.3)</code>	-5 6
<code>Max(float1, float2)</code>	The larger of 2 values		
<code>Min(float1, float2)</code>	The smaller of 2 values		
<code>Pi</code>	The ratio of the circumference of a circle to its diameter: π .		
<code>Pow(float1, float2)</code>	The power of <i>float1</i> to <i>float2</i>		
<code>Round(float)</code>	The rounded value of <i>float</i>		
<code>Sign</code>	The signing of a number: -1: <i>float</i> < 0 0: <i>float</i> == 0 1: <i>float</i> > 0		

Request

Request gives access to the HTTP request information.

Method	Description	Example	Result
Base	Base url of the request	<code>Request.Base</code>	'http://localhost:8080'
Cookies	Array of client cookies		
Get	Array of values from the query string		
Headers	Array of the HTTP headers of the request	<code>Request.Headers['host']</code>	'localhost:8080'
Post	Array of form values from the POST data. Currently only content type 'application/x-www-form-urlencoded' is supported.		
Query	Full query string of the request	<code>Request.Query</code>	'?cmd=test'
RawPost	String with the raw POST data.		
<code>SendCookie(name, value)</code>	Add or replace a cookie to/in the response		
<code>SendHeader(name, value)</code>	Add an HTTP header to the response		
Url	Url of the request	<code>Request.Url</code>	'http://localhost:8080/test.html'
User	Authenticated user name	<code>Request.User</code>	'admin'

Plugwise Objects

Plugwise

The Plugwise object is the root object of all the Plugwise system objects.

Method	Description	Example	Result
Appliances	Array of all the appliances	Plugwise.Appliances["TV"].Name	"TV"
ClassName	The class name of the object		
Groups	Array of all the groups		
ImagesPath	Virtual path to dynamic images		
Language	Current language code of application	Plugwise.Language	"nl"
Modules	Array of all the modules		
Rooms	Array of all the rooms		
Version	Application version of Source		

Appliance

The Appliance object is the representation of the 'Appliance' entity in the application.

All returned information is 'last known', not necessarily 'current'. This prevents page delays as a result of slow communication or offline modules. Immediately after the last known info is returned, a request to the application is queued to refresh the info, so that the next time the information is requested, an updated version is returned.

Method	Description	Example	Result
Appliance(<i>id</i>)	Constructor. Returns the appliance with id <i>id</i>	<code>\$id=Plugwise.Appliances[0].Id Appliance(\$id).SwitchOff()</code>	
ClassName	The class name of the object		
DoNotSwitchOff	True if the appliance is flagged not to switch off.		
Id	Internal ID of the appliance		
IsOff	True if the (module of the) appliance is switched off.		
IsOn	True if the (module of the) appliance is switched on.		
ImageName	Name of the virtual image file		
Module	Module to which the appliance is attached		
Name	Name of the appliance	Plugwise.Appliances["TV"].Name	"TV"
PowerState	Power state of the appliance: 'on' or 'off'		
PowerUsage	Last known power usage		
SwitchOn()	Switch the (module of the) appliance on		
SwitchOff()	Switch the (module of the) appliance off		
StatusImageName	Name of the virtual image that includes the status		
TotalUsage	Total power usage since the last counter reset		
Type	Appliance type		
TypeText	Appliance type translated to the current language		

Module

The Module object is the representation of the 'Module' or 'Plug' entity in the application.

All returned information is 'real time', so using the Module object can cause page delays, since execution of the template is halted until the requested information is received from the module.

Method	Description	Example	Result
Appliance	The assigned appliance		
ClassName	The class name of the object		
CloseRelay()	Close the relay; switch on the connected appliance		
Id	Internal ID of the module		
ImageName	Name of the virtual image file		
MacAddress	MAC address (hardware address) of the module.		
Name	Name of the module		
OpenRelay()	Open the relay; switch off the connected appliance		
PowerUsage	Last known power usage		
RelayState	Switch state of the relay: 'open' or 'closed'		
StatusImageName	Name of the virtual image that includes the status		
Status	Status of the module: 'online', 'offline' or 'unknown'		
Type	Module type id		
TypeText	Module type translated to the current language		

Group

The Group object is the representation of the 'Group' entity in the application.

Method	Description	Example	Result
Appliances	Array of appliances which are member of the group		
ClassName	The class name of the object		
Id	Internal ID of the group		
Name	Name of the group		
PowerUsage	Total of the appliances last known power usage		
SwitchOn()	Switch on the (modules of the) appliances assigned to the group		
SwitchOff()	Switch off the (module of the) appliances assigned to the group		
TotalUsage	Total power all the appliances usage since their last counter reset		
TotalUsageToday	Total power all the appliances usage for today		

Room

The Room object is the representation of the 'Room' entity in the application.

Method	Description	Example	Result
Appliances	Array of appliances which are assigned to the room		
ClassName	The class name of the object		
Id	Internal ID of the room		
Name	Name of the room		
PowerUsage	Total of the appliances last known power usage		

- Experimental and Preliminary -

SwitchOn()	Switch on the (modules of the) appliances assigned to the room		
SwitchOff()	Switch off the (module of the) appliances assigned to the room		
TotalUsage	Total power all the appliances usage since their last counter reset		
TotalUsageToday	Total power all the appliances usage for today		
Type	Room type id		
TypeText	Room type translated to the current language		

General remarks

Operator precedence

The engine does not (yet) support operator precedence; i.e. 'multiply' '*' normally has precedence over 'add' '+'. Instead expressions are evaluated from right to left. Use round brackets to assure the correct order in calculations.

Example	Result
<code>\$a=5+4*3</code>	17
<code>\$a=4*3+5</code>	32
<code>\$a=(4*3)+5</code>	17

Forms

When using HTML POST forms, you can combine form fields in an array by using square brackets in the field name:

```
<html><body><%
// set to posted values or an empty array
$cks=Request.Post.ContainsKey('ck')?Request.Post['ck']:{}
echo $cks // Show the contents of the array
$flds={'One','Two','Three'}
%><form method="POST" ><%
foreach $flds
  $v='chk_'+$_Index
  // keep the checkboxes checked that were checked by the user
%><%= $_Index%>
  <input type="checkbox" name="ck[]" value="<%= $v%>" <%= $cks.ContainsValue($v)?'
checked':''%>>
  <%= $_Value%><br><%
/foreach
%><input type="submit" Value="Submit">
</form>
</body></html>
```

You can also use keys. Note that here the keys do not require to be unclosed in quotation marks:

```
<html><body><%
// set to posted values or an empty array
$cks=Request.Post.ContainsKey('ck')?Request.Post['ck']:{}
echo $cks // Show the contents of the array
$flds={'1st'=>'One','2nd'=>'Two','3rd'=>'Three'}
%><form method="POST" ><%
foreach $flds
  // keep the checkboxes checked that were checked by the user
%><%= $_Index%>
  <input type="checkbox" name="ck[<%= $_Key%>]" value="<%= $_Value%>"
  <%= $cks.ContainsKey($_key)?' checked':''%>>
  <%= $_Value%><br><%
/foreach
%><input type="submit" Value="Submit">
</form>
</body></html>
```