Plugwise Template Engine

Titel	Plugwise Template Engine
Version	0.95
Date	2008-07-15
Product	Source/PTE
Author	TVR
Notes	This is a highly 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

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
```

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 the 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 an 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.	`$a={1}+{2,3}`
		`$a+={4,5}`
			`{0=>1,1=>2,2=>3}`
			`{0=>1,1=>2,2=>3,3=>4,4=>5}`

-
-=
	Remove one or more elements.
	If a key is given, the value is ignored.
		`$c=$a-{2,5}`
		`$b-={'Two'=>"Don't care"}`
			`{0=>1,1=>3,2=>4}`
			`{'One'=>'1'}`

==	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.
		`$a=={'1'}`
		`$a={3,1,2}`
		`$b={1,2,3}`
		`$a==$b`
			`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 *`key`*		
`ContainsValue(`*`value`*`)`	`True` if the array contains an element with value *`value`*		

`Count`	Number of elements	`$a={"abc",5,"xy"};`
		`$a.Count`
			
			3

`First`	First element	`$a.First`	`"abc"`
`GetUnique()`	Returns a copy of the array minus the duplicate elements		

`Join(sep)`	Concatenate all the values to one string using *sep* as separator.	`$a.Join(";")`	`"abc;5;xy"`
`Keys`	Array of all keys. For elements without a key, the index is returned.	`$b={'One'=>'1','Two'=>'2',7}` `$b.Keys`	`{'One','Two',2}`
`Last`	Last element	`$a.Last`	`"xy"`
`Values`	Array of all values.	`$b.Values`	`{'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
`==`	Is equal too	`$a==True`	`False`
`!=`	Is not equal to	`$a!=False`	
`!`	Logical NOT		
`&&`	Logical AND		
`||`	Logical OR		
`bool?expr1:expr2`	If *`bool`* equals `True` the result of the whole expression will be the result of *`expr1`*. Otherwise it will be the result of *`expr2`*.	`$f=4` `$s=($f==4)? "Yes" : "No"`	`"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
`+` `+=`	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.	`$d=DateTime();` `$d2=$d+DateTime("0010-01-01");` `$d2+=3600;`	`"2008-06-11 16:28:38"` `"2017-06-11 16:28:38"` `"2017-06-11 17:28:38"`
`-` `-=`	Subtract a date or a number of seconds. See '+'.	`$d-=DateTime("12:00:00");`	`"2008-06-11 04:28:38"`
`==`	Is Equal to.	`$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	$s="a"+"b" $s="4"+5 $s=4+"5" $s+="a"	"ab" "45" 9 "45a"
- -=	Remove all occurrences of the second string from the first.	$s="Hello World"-"l" $s-="o"	"Heo Word" "He Wrd"
==	Is equal too	"ab"=="ab"	False
!=	Is not equal to	"ab"!="ab"	True
>	Greater than	"ac">"ab"	True

<	Less than	`"ac"<"ab"`	`False`
>=	Greater than or equal to	`"ab">="ab"`	`True`
<=	Less than or equal to	`"ac"<"ab"`	`False`

`*`
`*=`
	Concatenate a string multiple times	`$s="-"*4`
		`$s*=2`
			`"----"`
			`"--------"`

`[index]`	The character at position *index*	`$s="abcdef"`
		`$s[3]`
			`"d"`

Member	Description	Example	Result
`ClassName`	The class name of the object		

`IndexOf(string)`	The zero based start position of the first occurrence of *string*	`$s="Hello world";`
		`$s.IndexOf("l");`
			2

`LastIndexOf(string)`	The start position of the last occurrence of *string*	`$s.LasIndexOf("l");`	9
`Length`	The length	`$s.Length`	11
`Lower`	The lower case version	`$s.Lower`	`"hello world"`
`MD5`	The MD5 hash of the string		
`Replace(string1, string2)`	Replaces each occurrence of *string1 with string2*	`$s.Replace("o","0")`	`"Hell0 w0rld"`

`Split(string [,int])`	Split a string on separator *string* to an optional maximum of *int*	`$s.Split("l");`
		
		`$s.Split("l",2);`
			`{0=>'He',1=>'',`
			`2=>'o wor',3=>'d'}`
			`{0=>'He',1=>'lo world'}`

`Substring(int1 [,int2])`	The string part starting from *int1* optionally with a maximum length of *int2*	`$s.Split(6);`
		`$s.Split(6,2);`
			`"world"`
			`"wo"`

`Trim()`	Remove white spaces from beginning and end of string	`"  Hello\n".Trim()`	`"Hello"`
`Upper`	The upper case version	`$s.Upper`	`"HELLO WORLD"`
`UrlDecode()`	Decodes the URL endode 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**

`<%="Hello world" %><br>`
`<% $a=5 %>`
`<%=$a%>
`
	`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**

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

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**

`<%`
` Echo "Hello world!"`
`%>`
	`Hello world!`

Exit

```
Exit [string]
```

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

Example	**Output**

`<%`
`  Echo "Hello world!"`
`  Exit;`
`  Echo "This is not shown"`
`%>`
	`Hello world`

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**

`<%`
`$a={'1'=>'One','2'=>'Two','3'=>'Three','4'=>'Four'}`
`ForEach $a`
`  if $_Index==1`
`    continue`
`  /If`
	`$a[0] = {1=>One}`
	`$a[2] = {3=>Three}`

`%>`	
`$a[<%=$_Index%>] = {<%=$_Key%>=><%=$_Value%>}<br>`	
`<%`	
`if $_Value=='Three'`	
`    break;`	
`  /If`	
`/Foreach`	
`%>`	

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

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
`<%`	`$a is Three`
` $a=3;$b=1`	`$b is One`
` Echo "$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
%>
```

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

```
<%
$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]%>}<br>
<%
  If $a[$ix]=='Three'
    break;
  /If
/While
%>
```

Output

```
$a[0] = {One}
$a[2] = {Three}
```

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

```
<%
 $a={'d'}
 $b={'a', 'c', 'd'}
 With $a
  Echo .Count,"<br>"
 /With
 With $b
  Echo .Count,"<br>"
 /With
%>
```

Output

```
1
3
```

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**

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

Engine objects

System

System is the main object of the template engine.

Method	Description	Example	Result
`Blocks`	Array of all the defined blocks	See **`Write`**	
`Date`	String with current local date	`System.Date`	`16-06-2008`
`Path`	Local path to the server root folder	`System.Path`	`C:\Program Files\Plugwise\Plugwise Source\ www`
`Settings`	Array with all the name-value pairs as specified in the ini file under the [Settings] category.		
`Time`	String with current local time	`System.Time`	`21:37:33`
`Version`	Version string of the engine	`System.Version`	`0.9`

Math

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

Method	Description	Example	Result
Abs(*float*)	The absolute value of *float*	`$d=Math.Abs(-5);`	5

Ceil(*float*)	The smallest integer greater than or equal to *float*	`Math.Ceil(-5.3)`
		`Math.Ceil(5.3)`
			-5
			6

E	The natural logarithmic base e		

Floor(*float*)	The largets integer less than or equal to *float*	`Math.Ceil(-5.3)`
		`Math.Ceil(5.3)`
			-5
			6

Max(*float1*, *float2*)	The larger of 2 values		
Min(*float1*, *float2*)	The smaller of 2 values		
Pi	The ratio of the circumference of a circle to its diameter: π.		
Pow(*float1*, *float2*)	The power of *float1* to *float2*		
Round(*float*)	The rounded value of *float*		
Sign	The signing of a number: -1: *float* <0 0: *float*==0 1: *float*>0		

Request

Request gives access to the HTTP request information.

Method	Description	Example	Result
`Base`	Base url of the request	`Request.Base`	`'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	`Request.Headers['host']`	`'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	`Request.Query`	`'?cmd=test'`
`RawPost`	String with the raw POST data.		
`SendCookie(`*name*`,` *value*`)`	Add or replace a cookie to/in the response		
`SendHeader(`*name*`,` *value*`)`	Add an HTTP header to the response		
`Url`	Url of the request	`Request.Url`	`'http://localhost:8080/test.html'`
`User`	Authenticated user name	`Request.User`	`'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	`<img src="<%=Plugwise.ImagesPath%>32/<%=.ImageName%>.png">`	`<img src="/pwimg/32/appliance.png">`
`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(`*`id`*`)`	Constructor. Returns the appliance with id *`id`*	$id=Plugwise.Appliances[0].Id Appliance($id).SwitchOff()	
`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	`<img src="<%=Plugwise.ImagesPath%>32/<%=.StatusImageName%>.png">`	`<img src="/pwimg/32/appliance_on.png">`
`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 the connected appliance on		
`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 the connected appliance off		
`PowerUsage`	Last known power usage		
`RelayState`	Switch state of the relay: ‘open’ or ‘closed’		
`StatusImageName`	Name of the virtual image that includes the status	`<img src="<%=Plugwise.ImagesPath%>32/<%=.StatusImageName%>.png">`	`<img src="/pwimg/32/appliance_on.png">`
`Status`	Status of the module: ‘online’, ‘offline’ of ‘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		

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		

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
`$a=5+4*3`	17
`$a=4*3+5`	32
`$a=(4*3)+5`	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>
```