Plugwise Script Engine

Title	Plugwise Script Engine
Version	2.23
Date	2012-12-09
Product	Source/PTE
Author	TVR
Notes	This is an unsupported 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
	2.10: CRUD functionality for existing and new Plugwise classes
	2.11: 'ELSE IF' is treated as 'ELSEIF'.
	    A '\n' in a script file is always treated as a EOLN, with or without a '\r'.
	    Implementation of functions (blocks are considered obsolete now)
	    Functions can also be used to add custom methods to existing classes.
	2.12: Support for sessions through nonvolatile array 'Request.Session'.
	    RegExp en Http classes added
	    Apache style access logging
	    AutoScript and AutoScriptInterval for periodically automated scripts
	    File extension .PTE is supported so editors can recognize script files
	2.13: Added to Array: Avg(), Sum(), Max(), Min(), Sort(), SortByKey(),
	    Remove(), RemoveAt(), RemoveByKey()
	    Log() added to Appliance, Group and Room
	    Multi line comments using /* and */
	    Added to DateTime: AddSeconds(), AddMinutes(), AddHours(),
	    AddDays(), AddMonths(), AddYears()
	2.14: Added to Module: FirmwareDate, FirmwareVersion, HardwareVersion
	    Added to Network: GetModuleList()
	    Added to System: LanAdapters
	2.15: Web server is now multi-threaded
	    Changed /sys/mimetypes.txt so html output is considered to be utf-8.
	    Icons of /pwimg/ are transparent PNG's
	    Power usage graphs can be generated via /pwgraph/
	    Added to System: Execute(), ResetTimer()
	    Added PowerState to Group and Room
	    Added SetSchedule() to Appliance, Group and Room
	    Added to Plugwise: Currency, FeatureFlags, License, PersonalInfo, Register(), Restart(), ScanPorts(), SetLicense(), SetPersonalInfo()
	    Added PeakDaysOfWeek and SetPeakDaysOfWeek() to Tariff
	2.16: New type 'Undefined' added.
	    Changed type of unexisting array element to 'Undefined' instead of an empty string.
	    Added || operator to Array, DateTime, Float and String for default value assignments
	    Added System.SetCompatibility()
	    Added 'emptyelement' compatibility flag
	    Added optional *TariffType* parameter to .Log()
	    Added To Plugwise: .LogData(), .ColorScheme(), SetColorScheme()
	    Support for hexadecimal values like '0x80ff80'
	    Documentation /pwimg/ and /pwgraph/
	    Added Network.Quality
	2.17: Added Module.Temperature, Module.Humidity and Room.TemperatureAndHumidity for Sense.
	2.20: Implemented basic url rewriting with _catch404.pte
	    Added Request.Method
	    Added Group.SetBroadcast
	    Added Group.Type, Group.Typetext, Group.SetType()
	    Added Group.SetBroadcast()
	    Added .UUId to all Plugwise objects

	Added Trigger object Added Plugwise.Backup() Added Module.LastTelegram Added .SetExtra and .GetExtra() 2.22: Implemented event scripts (experimental) More informative error handling, including function call traceback Improved array element handling. Can result up to 80% speed gain. Renamed .GetExtra() and .SetExtra to .GetPersistent() and .SetPersistent() Added .Runtime[] member to all Plugwise objects Added Trigger.EventCount Added Array operators: '&', '|' and '^' Added Array.Sort(*callback*), Array.Filter("*expression*") and Array.Filter(*callback*) 'index.xml' is added to default pages (last in search order) String.SubString(): length parameter can be negative to be relative to current length Added optional 'As $*alias*' to ForEach Added Library *file* to include function libraries Added Climate, Occupation and OnOff graphs Rounding method changed to 'Nearest' (0.5-->1) instead of .NET's default 'Bankers' (0.5-->0) 2.23: Added String.Replace(*array*) Added Plugwise.SendNotification() Finalization event scripts, added chapter on events.

Inhoud

Introduction

The Plugwise Source application has a built in lightweight multi-threaded web server with a simple object oriented script 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.

From version 2.1, full CRUD is supported, so you can change the configuration of the system via scripting.

Note: Whether the web server is available and which functionality is enabled depends on the license type of your Source application.

> The script engine is also used outside the webserver to support events and notifications.

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

; This script should be executed every 5 seconds
autoscript=dispatcher.pte
autoscriptinterval=5

[settings]
; Here you can specify your own configuration 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 also 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.
Note: Source and PlugwiseServer cannot run at the same time.

The Basics

Any file requested by a client (i.e. web browser) that has one of the extensions '.css', '.html', '.htm', '.txt', '.xml' or '.pte' 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.pte', 'index.html' or 'index.xml' in that search order.

Handling of 404(_catch404.pte)

If a requested url does not exists a 404-Page-not-found error is returned unless the server finds a script called '_catch404.pte'. The server will try to find this file as deep in (the valid part of) the requested path as possible and then up to the root. If it finds it, the script is executed and the resulting url is handled instead of the requested.
Inside the script the original url is stored in $_script and the script should change this variable to change the url to handle.

```
<%
 $_script="index.html" // redirect any 404 to the index page
%>
```

Except for session variables all variables are local and cannot be passed to other scripts.

Debugging

The script engine has very limited debugging features. When using functions, it will dump a trace back of the function call, but the line numbers are relative to the function body, not to the script file.
Using the /tracehttpd switch on the command line of Source will open a window that shows all the http calls to the webserver.

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.
All variables are treated as objects although there is a 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 (their 'value').

```
<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.

For DateTime, Float and String variables the '||' operator (logical OR) has a special function: If the left value is Undefined, then the right value is used and the left value is ignored, otherwise the left value is used and the right value is ignored.

So instead of

```
$param=Request.Get["myparam"]
if $param == undefined
  $param='some default'
/if
```

You can use

```
$param=Request.Get["myparam"] || 'some default'
```

Casting

To assign a value to a variable of a different type for example a float to a (formatted) string you can use casting.

```
<%
  $f=12.345
  echo $f, ', ', String($f), ', ', String($f,'0.00'), '<br>'
  $d= DateTime('2007-06-01')
  echo $d, ', ', String($d), ', ', String($d,'yyyy MMM d'), '<br>'
  exit
%>
```

Result (depends on Windows' language and region settings):

```
12.345, 12.345, 12,35
2007-06-01 00:00:00, 2007-06-01 00:00:00, 2007 jun 1
```

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 by assigning a single element:

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

The default for a nonexistent array element, is an `Undefined` value[*]. Use curly brackets or Array.Fill() to preset array elements to other types and values.

```
$arr={0}; $arr[0]+=1; $arr[0]+=2;
echo $arr,'<br>'
$arr={}.Fill(0,1); $arr[0]+=1; $arr[0]+=2;
echo $arr,'<br>'
```

Outputs:

```
{ 3 }
{ 3 }
```

Existing named elements can also be addressed by using '.*key*' instead of '["*key*"]'. Again, the name of the key is case insensitive.

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

Outputs:

```
5
```

Operator	Description	Example	Result
+ +=	Add one or more elements.	$a={1}+{2,3} $a+={4,5}	{1,2,3} {1,2,3,4,5}
[]	Add one elements to the end of the array	$a={1,3} $a[]=7	{1,2} {1,2,7}

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

`&`
`&=`
	The elements that exists in both arrays.	`$a={1,2,3,4,5}`
		`$b={4,5,6,7,8}`
		`Echo $a & $b`
			`{4,5}`	

`|`
`|=`
	The elements in both arrays combined. Duplicates are skipped	`$a={1,2,3,4,5}`
		`$b={4,5,6,7,8}`
		`Echo $a
			$b`	`{1,2,3,4,5,6,7,8}`	

`^`
`^=`
	The elements in either array, but not in both array.	`$a={1,2,3,4,5}`
		`$b={4,5,6,7,8}`
		`Echo $a ^ $b`
			`{1,2,3,6,7,8}`	

`==`	Is Equal to.
	Two arrays 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 and the order of the values are ignored.
		`$a=={'1'}`
		`$a={3,1,2}`
		`$b={1,2,3}`
		`$a==$b`
			`False`
			
			`True`	

`!=`	Is not equal to, reverse of '=='			

`||`	Unless the left operant is Undefined, ignore the right operant, otherwise ignore the left operant.	`$a = {{'a','b'}}`
		`$d = $a[0] || {'x','y'}`
		`$d = $a[2] || {'x','y'}`
			`{'a','b'}`
			`{'x','y'}`	

Member	Description	Example	Result
`Avg()`	The average of all the floats in the array	`{9,4,"xy",8}.Avg()`	`7`
`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`
`Fill(value, count)`	Fills the array with `count` (copies of) `value`. Existing elements are removed.	`$a.Fill(1,5).Count`	`5`

`Filter(callback [, data])`	Call the function *callback* for each element. Each element on which the function returns True is copied to the resulting array. If the result is not a Boolean then the result is copied to the resulting array.
	The key is passed through the first and the value through the second parameter of the function. The optional parameter *Data* is the third.
		```<% Function MyFilter($key,$value,$data) %>```
		```<%```
		``` return $value==$data```
		```%>```
		```<% /Function %>```
		```<% Function Filter2($key,$value) %>```
		```<%```
		``` return $key+": "+String($value)```
		```%>```
		```<% /Function %>```
		```<%```
		```$b={"x"=>5,"y"=>"three","b"=>12,"c"=>5,"g"=>"gee"}```
		```echo $b.Filter(MyFilter,5)```
		```echo "\n"```
		```echo $b.Filter(Filter2)```
		```echo "\n"```
		```echo $b.Filter("$_value==12")```
		```echo "\n"```
		```echo $b.Filter("$_key")```
		```%>```	

`Filter("expression")`	Evaluates the expression *"expression"* for each element. Each element on which the result equals True is copied to the resulting array. If the result is not a Boolean then the result is copied to the resulting array.
	The key is passed as $_key and the value as $_value.
		`{ 'x'=>5, 'c'=>5 }`
		`{ 'x'=>'x: 5', 'y'=>'y: three', 'b'=>'b: 12', 'c'=>'c: 5', 'g'=>'g: gee' }`
		`{ 'b'=>12 }`
		`{ 'x'=>'x', 'y'=>'y', 'b'=>'b', 'c'=>'c', 'g'=>'g' }`	

`First`	First element	`{"abc",5,"xy"}.First`	`"abc"`
`IndexOf(value)`	Zero based index of the `value` in the array. If the array does not contain the element, the result will be -1.		
`GetUnique()`	Returns a copy of the array minus the duplicate elements	`{"abc",5,"xy",5}.GetUnique()`	`{"abc",5,"xy"}`
`Join(sep)`	Concatenate all the values to one string using *sep* as separator.	`{"abc",5,"xy"}.Join(";")`	`"abc;5;xy"`
`Keys`	Array of all keys. For elements without a key, the index is returned.	`{'One'=>'1','Two'=>'2',7}.Keys`	`{'One','Two',2}`
`Last`	Last element	`{"abc",5,"xy"}.Last`	`"xy"`

Max()	The largest of all floats in the array.	`{2,4,"xy",8}.Max()`	8
Min()	The smallest of all floats in the array.	`{2,4,"xy",8}.Min()`	2
Remove(*value*)	Removes any element from the array that has a value equal to *value*. The result is the array itself.		
RemoveAt(*index*)	Removes the element at position *index* in the array. The result is the array itself.		
RemoveByKey(*key*)	Removes the element that has the string *key* as key. The result is the array itself.		
RemoveNulls()	Removes all elements that have a `null` value. The result is the array itself.		
Reverse()	Reverses the order of elements in the array. The result is the array itself.		
Sort()	Sorts the array by the values. The result is the array itself.	`$a= {'a'=>2,'c'=>6,1,'g'=>8,'i'=>76,'h'=>5,0,5,'b'=>4}` `echo $a,' ';` `{ 'a'=>2, 'c'=>3, 1, 'g'=>8, 'i'=>76, 'h'=>5, 0, 5, 'b'=>4 }` `echo $a.Sort(),' '` `{ 0, 1, 'a'=>2, 'c'=>3, 'b'=>4, 5, 'h'=>5, 'g'=>8, 'i'=>76 }`	
Sort(*subkey*)	The array members are expected to be arrays too and the sorting is based on their values for *subkey*. If a member is not an array then its (single) value is used in the sort.	`$a={'a'=>{'i'=>2,'j'=>8},'c'=>{'i'=>12,'j'=>18},1,'g'=>{'i'=> 9,'j'=>6},'i'=>{'i'=>4,'j'=>20},'h'=>5,'b'=>{'i'=>1,'j'=>12}}` `echo $a.Sort('i'),' '` `{ 1, 'a'=>'{ 'i'=>2, 'j'=>8 }', 'i'=>'{ 'i'=>4, 'j'=>20 }', 'h'=>5, 'c'=>'{ 'i'=>12, 'j'=>18 }' }` `echo $a.Sort('j'),' '` `{ 1, 'h'=>5, 'a'=>'{ 'i'=>2, 'j'=>8 }', 'c'=>'{ 'i'=>12, 'j'=>18 }', 'i'=>'{ 'i'=>4, 'j'=>20 }' }`	

Sort(*callback*)	*Callback* is the name of a function with 2 parameters that is called to compare 2 elements to determine their sorting order in the array. If the first precedes the second in sorting order, the function should return -1. If the second precedes the first it should return 1 If they are equal it should return 0.	`<% Function MySort($e1,$e2) %>`
		`<%`
		`  return $e1-$e2`
		`%>`
		`<% /Function %>`
		`<%`
		`$a={4,2,6,8,5}`
		`$a.Sort(MySort)`
		`echo $a`
		`{ 2, 4, 5, 6, 8 }`	

SortByKey()	Sorts the array by the keys. The result is the array itself.	`echo $a.SortByKey(),' '` `{ 0, 1, 5, 'a'=>2, 'b'=>4, 'c'=>3, 'g'=>8, 'h'=>5, 'i'=>76 }`	
Sum()	The sum of all floats in the array.	`{9,4,"xy",8}.Sum()`	21
Values	Array of all values.	`{'One'=>'1','Two'=>'2',7}.Value s`	`{'1', '2',7}`

* In all versions before 2.16 a nonexistent array element returned an empty string. Backwards compatibility can be assured with the '`compatibility=emptyelement`' application flag or with a '`System.SetCompatibility('emptyelement',True)`' call. See also the System object.
This behavior can also be mimicked using the '||' (logical OR) operator. See the Variables section and the operator tables of the Array, String, Float or DateTime types.

Bool

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

Operator	Description	Example	Result
==	Is equal too	`$a=False;` `$a==True`	`False`
!=	Is not equal to	`$a!=False`	`True`
!	Logical NOT		
&&	Logical AND		
||	Logical OR		
^	Exclusive 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*. Note: Because the engine lacks operator precedence you must enclose the *bool* expression with round brackets.	`$f=4` `$s=($f==4)? "Yes" : "No"`	`"Yes"`

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

DateTime

A DateTime is an 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`

||	Unless the left operant is Undefined, ignore the right operant, otherwise ignore the left operant.	`$a = {DateTime("2008-06-10")}`
		`$d = $a[0] || DateTime()`
		`$d = $a[2] || DateTime()`
			`"2008-06-10 00:00:00"`
			`"2008-06-11 16:28:38"`

Member	Description	Example	Result

`DateTime()`
`DateTime(string)`
`DateTime(seconds)`
	The current date and time
	Casts the string to a date
	Casts the float *seconds* to a date		

`AddDays(days)`	Adds a number of days to the date.		
`AddHours(hours)`	Adds a number of hours to the date.		
`AddMinutes(minutes)`	Adds a number of minutes to the date.		
`AddMonths(months)`	Adds a number of months to the date.		
`AddSeconds(days)`	Adds a number of seconds to the date.		
`AddYears(days)`	Adds a number of years to the date.		
`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=$d.Day;`	`11`
`Format(format)`	Formats the date to the given format. '*format*' syntax is according to .Net DateTime object.	`Echo DateTime().Format("yyyyMMdd HHmmss");`	`20080611162838`
`Hour`	The hour of the day	`$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"
UTCSeconds	The UTC equivalent in seconds since 1-1-1970 (Unix epoch)	`$utcsec=$dd.UTCSeconds`	1213187318
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. Hexadecimal numbers must be preceded by '0x', '0xff' equals '255'. To output a Float in hexadecimal format use String(*float*,"x").

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	`$f=63&0x11` `$f&=8`	17 0
| |=	Binary OR	`$f=0x87\|14` `$f|=18`	143 159
^ ^=	Binary exclusive OR (XOR)	`$f=15^7` `$f^=15`	8 7

||	Unless the left operant is Undefined, ignore the right operant, otherwise ignore the left operant.	`$a = {12}`
		 `$d = $a[0] || 5`
		 `$d = $a[2] || 5`
			
			 12
			 5

Member	Description	Example	Result
`Float(string)`	Casts a string to a float		
`ClassName`	The class name of the object		
`Round([decimals])`	Rounds the float to an optional number of decimals. Default is no decimals.		
`String(format)`	Converts the float to a string using the specified format string.		

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. Comparisons between strings are case insensitive. When using double quotes special characters can be escaped using the back slash '\', i.e. \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'`	`True`
!=	Is not equal to	`"ab"!="ba"`	`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*. If *index* is negative, the position is relative to the end of the string.	`$s="abcdef"`
		`$s[3]`
		`$s[-1]`
			
			`"d"`
			`"f"`

||	Unless the left operant is Undefined, ignore the right operant, otherwise ignore the left operant.	`$a = {"abcdef"}`
		`$d = $a[0] || "yxz"`
		`$d = $a[2] || "yxz"`
			
			`"abcdef"`
			`"yxz"`

Member	Description	Example	Result
`String(value)` `String(value[,format])`	Casts a value to a string Casts a value to a string using the specified format string.		
`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*	`$s.LastIndexOf("l")`	`9`
`Length`	The length	`$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"`
`Replace(array)`	Replaces each occurrence of a key with the value of each key/value pair of *array*		

`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*. If *int1* is negative then the start is relative to the end of the string. If *int2* is negative then the requested length is relative to the current length.	`$s.Substring(6)`
		`$s.Substring (6,2)`
		`$s.Substring (-4,2)`
		`$s.Substring (0,-2)`
			`"world"`
			`"wo"`
			`"or"`
			`"wor"`

`Trim()`	Remove white spaces from beginning and end of string	`"  Hello\n".Trim()`	`"Hello"`
`Upper`	The upper case version	`$s.Upper`	`"HELLO WORLD"`
`UrlDecode()`	Decodes the URL encoded string	`$a="Hello <World>"` `echo $a.UrlEncode()`	`Hello+%3cWorld%3e`
`UrlEncode`	URL encodes the string	`Echo "Hello+%3cWorld%3e" .UrlDecode()`	`Hello <World>`

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

Block, /Block

*** Obsolete. Use 'Function' instead ***

```
<% 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 prevents 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!"`	`Hello world`

```
    Exit;
    Echo "This is not shown"
%>
```

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

```
ForEach array
        Loop
/ForEach
ForEach array As $name
        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. If an alias is specified with **As** $*name*, then that alias has the same value as $_Value.

Example	Output

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

Format

```
Format name As 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 C# method **String.Format()** are used.

Example	Output

<pre><%
  $a={'a', 'c', 'd'}
  $b=1.574
  $f=1.574
 	
  Echo "$a.Count=",$a.Count,"
"
  Echo "$b=",$b,"
"
  Echo "$f=",$f,"
"
 	
  Format "Float.f" As "{0:0.0}"</pre>
	<pre>$a.Count=3
	$b=1.574
	$f=1.574
	$a.Count='3'
	$b=1,57
	$f=1,6</pre>

```
  Format "Float" As "{0:0.00}" // All other floats!
  Format "Array.Count" As "'{0}'"

  Echo "$a.Count=",$a.Count,"<br>"
  Echo "$b=",$b,"<br>"
  Echo "$f=",$f,"<br>"
%>
```

Function [Return], /Function

```
<% Function name([argument1, …]) %>
        …
        [Return [expression]]
<% /Function %>
```

```
Function name([argument1, …])
        …
        [Return [expression]]
/Function
```

Defines a script part (function) that can be called from anywhere in the script as a statement or as (part of) an expression. A function can contain anything except another function definition. Within a function other functions and the function itself (recursion) can be called. Unless a function is defined in a library file, **Function** and /**Function** must be enclosed with their own tags. Use **Return** to exit a function and optionally pass a value to the calling expression. More than 1 **Return** statement can be used in the function body. Overloading is supported, which means you can define 2 or more functions with the same name as long as their number of arguments are different.

Variables within a function are always local; they are destroyed when the function exits. Also, variables outside the function are not accessible inside the function.

Example	Output

<pre><% **function** Factorial($v1) %>
<% *// Recursion example*
  if $v1==0
    return 1
  /**if**
  return $v1 * Factorial($v1-1)
%>
<% /**function** %>
<% **function** ShowFactorial($v1) %>
<% *// No result, just output*
  echo '6! = ',Factorial(6),'
'
%>
<% /**function** %>
<%
  // Used as a statement
  ShowFactorial($v1)
%>
<% **function** Add($v1,$v2) %>
<% *// simple function*
  return $v1+$v2
%>
<% /**function** %>
 	
<% **function** Add($v1,$v2,$v3) %>
<% *// overloading example*
  return $v1+$v2+$v3
%>
<% /**function** %>
 	
<% **function** Devide($v1,$v2) %>
<% *// termination example*
  if ($v2==0)
    exit "Devision By zero!"
  /**if**
  return $v1/$v2
%>
<% /**function** %>
 	
<%
  echo '7 + 3 = ', Add(7,3), '
'
  echo '7 + 3 + 5 = ', Add(7,3,5), '
'
  echo '7 / 3 = ', Devide(7,3), '
'
%></pre>
	<pre>6! = 720
	7 + 3 = 10
	7 + 3 + 5 = 15
	7 / 3 = 2.33333333333333</pre>

A function can also be used to add custom methods to existing classes by preceding the function name with the class name and a period '.'

When the function (i.e. method) is called, the subject (the object) of the method is accessible through the '$this' variable.

Example	Output

File: array.pte
 	
``
function Array.Avg2()
   // Custom method example.
   // This is a simulation of
   // the built-in Avg() method
   $sum=0;
   $cnt=0;
   foreach ($this)
    if $_value.ClassName=="float"
       $sum+=$_value
       ++$cnt
     /if
   /foreach
   if $cnt==0
     Return Null
   /if
   return $sum/$cnt
/function
``
 	
File: index.html
 	
``
   <%
   Library "array.pte"
   $a={1,2,3,4,5,6,7,8,9,10}
   echo "The average is ", $a.Avg2(), '
'
   %>
``
	The average is 5.5

If, [ElseIf | Else If], [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 and can also be written as **Else If**. 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

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

Include, Library

```
include path
library path
```

Include includes the file *path* into the current page. The code in the include file is processed and executed as though it is part of the current page. This is especially useful for commonly used script parts like initialization and format definitions. In each file the code must be surrounded with code tags.

Library differs from include in that it only processes the file and extracts the function definitions. Is does not execute the code in the file. The code in a library does not require code tags.

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[3] = {Four}
	$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>
	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

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

Engine objects

File
Static object for common file functions.

Method	Description	Example	Result
AppendLine(*path, string*)	Adds a line to the end of a file. CR and LF characters are added. If the file does not exist, it is created.		
CreatePath(*path*)	Creates all the directories in *path*. Returns True if successful, False otherwise.		
Date(*path*)	Last modification date of a file		
Delete(*path*)	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(*path*)	Returns True if the file exists, False otherwise.		
IsDirectory(*path*)	True if an existing directory		
IsFile(*path*)	True if an existing file		
Move(*path, destination*)	Move or rename a file or directory. *destination* must be the full path to the new name. If *destination* exists, it is deleted first. Returns True if successful, False otherwise.		
Read(*path*)	Reads the contents of a text file into an array; one line per element. The CR and/or LF characters are trimmed.		
Size(*path*)	The length in bytes of a file		
Write(*path, array*)	Writes an array to a file. One line for each element. CR and LF characters are added.		

Http
Http is used to retrieve (remote) web pages or data.

Method	Description	Example	Result
Get(*url*)	Returns the result of a HTTP-GET request to *url*		
Get(*url, data*)	Sends the array *data* as form data in a HTTP-POST request to *url* and returns the result.		
DoRequest(*url, method, contenttype, data*)	Sends an HTTP request to the url using given http method and content type.		
UrlEncode(*data*)	URL-encodes a string		
UrlDecode(*data*)	Decodes an URL-encoded string		

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 largest 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		

RegEx

RegEx enables the use of regular expressions.

Method	Description	Example	Result
Match(*expr, subject*)	Matches the regular expression *expr* on the string subject and returns the first match as an array. The first element contains the full match, the following elements contain the sub matches, if there were any.		
Matches(*expr, subject*)	Similar to Match(), but returns all the matches.		

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		
`Session`	Non volatile array that can be used to pass data between requests of the same client (session). Sessions expire when Source terminates		
`Url`	Url of the request	`Request.Url`	`'http://localhost:8080/test.html'`
`User`	Authenticated user name	`Request.User`	`'admin'`

System

System is the main object of the template engine.

Method	Description	Example	Result
`Blocks`	Array of all the defined blocks	See **Write**	
`Compatibility(`*string*`)`	Returns the value of a compatibility flag. Flags can be: - 'EmptyElement': nonexisting array elements return an empty String instead of a Undefined.		
`SetCompatibility(`*string, bool*`)`	Set or clear a compatibility flag		
`DataFolder`	Local path to the application data folder	`System.DataFolder`	`C:\Documents and Settings\me\Application Data`
`Date`	String with current local date	`System.Date`	`16-06-2008`
`EnvVars`	Array of the systems environment variables		
`Execute(`*program,* `[`*arguments,* `[`*directory*`]])`	Starts a program on the computer where Source is running. Note: If you a start a program that requires administrator rights, the computer locks up with a message box, that requires user interaction.	`System.Execute(` `"cmd.exe",` `"/c \a")`	*Sounds a beep.*
`LanAdapters`	Array with info about the networks adapters of the PC	`System.LanAdapters[0]`	`{ 'MACAddress'=>'00:1d:09:42:10:47', 'IP6Address'=> 'fe:80:00:00:00:00:00:00:ad:bc :15:14:cc:3c:12:f3', 'IPAddress'=>'10.0.2.138', 'IPMask'=>'255.255.255.0', 'Gateway'=>'10.0.2.254', 'Name'=>'LAN-verbinding', 'Description'=>'Broadcom NetXtreme 57xx Gigabit Controller', 'Type'=>'Ethernet' }`
`Path`	Local path to the server root folder		`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.		
`TempFolder`	Path to the temporary files folder	`System.TempFolder`	`C:\Documents and Settings\me\Local Settings\Temp`
`Time`	String with current local time	`System.Time`	`21:37:33`
`Version`	Version string of the engine	`System.Version`	`2.1`

Plugwise Objects

Note: An asterix ('*') in the first column means that that functionality is only available in the Pro version.

Plugwise

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

	Method	Description	Example	Result
	`Appliances`	Array of all the appliances with their Id as key.		

	Name	Description	Example	Result
	ApplianceTypes	Array of all the appliance types known to Source. Each type is defined in a array in the format: *types*[*type*]={ 'Title'=>*translated name*, 'Image'=*image name*}	`Echo Plugwise.ApplianceTypes['toaster']`	`{ 'Name'=>'toaster', 'Title'=>Toaster', 'Image'=>'toaster' }`
*	Backup(*path*)	Saves a backup of the current database to the specified path.		
	ClassName	The class name of the object		
	ColorScheme	Returns the current color scheme for graphs or the default if none is set.		
	SetColorScheme(*array*)	Sets the color scheme for graphs. *Array* does not need to contain all colors, just the ones you want to change from the default. Use Null to reset to default.	`Plugwise.SetColorScheme( {"background"=>0x004000})` `Plugwise.SetColorScheme(Null )`	
*	CreateAppliance(*name*)	Creates a new appliance		
*	CreateGroup(*name*)	Creates a new group		
*	CreateModule(*name*)	Creates a new module		
*	CreateNetwork(*name*)	Creates a new network		
*	CreateRoom(*name*)	Creates a new room		
*	CreateSchedule(*name*)	Creates a new schedule		
*	CreateTariff(*name*)	Creates a new tariff		
	Currency	The used currency symbol in Source	`Echo Plugwise.Currency`	€
	DayCodes	Array of the short week day codes, used for schedules.	`Echo Plugwise.DayCodes`	`{ 0=>'su' 1=>'mo', 2=>'tu', 3=>'we', 4=>'th', 5=>'fr', 6=>'sa' }`
	FeatureFlags	The licensed features of Source	`Echo Plugwise.FeatureFlags`	`{'W','X'}`
	Groups	Array of all the groups with their Id as key.		
	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	`Echo Plugwise.Language`	Nl
*	LanAdapters	Array of all the active LAN adapters of the system		
	License	The product license string		
*	Logdata(*array*, *startdate* [, *enddate* [, *tarifftype*]])	Returns an array with the log data of type *tarifftype* of the appliances in *array* for the specified date or period. *tarifftype* can be 1 for usage or 257 for production. Default is 1		
*	SetLicense(*string*)	Sets the license with the given key. No other license may be active and the given key must be valid. Result: True if the new license is valid.	`SetPersonalInfo({ 'FirstName'=> 'Fred', 'LastName'=>'Flintstone' })`	
*	SendNotification(*recipient, subject, message [,type]*)	Uses the Plugwise Notification Service to send an email message to *recipient.*		
	Modules	Array of all the modules with their Id as key.		
	Networks	Array of all the networks with their Id as key.		
	PersonalInfo	Array of all personal info settings		
*	SetPersonalInfo(*array*)		`SetPersonalInfo({ 'FirstName'=> 'Fred', 'LastName'=>'Flintstone' })`	
*	Register()	Registers current license, personal info and modules at Plugwise server.		
	Restart()	Registers current license, personal info and modules at Plugwise server.		
	Rooms	Array of all the rooms with their Id as key.		
	Runtime	Array of user defined values that exists as long as Source is running.		
	ScanPorts([*array*])	Scan the given ports for Plugwise Stick. If no array is given, all COM ports are scanned. Result: array of found Networks.		
	Schedules	Array of all the schedules with their Id as key.		
	Tariffs	Array of all the tariffs with their Id as key.		
	SetUseUUids(*bool*)	Enables or disables the use of an object's UUid as array key instead of an integer.		
	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.

An existing appliance object can be obtained in 3 different ways

```
$app = Appliance(name)
$app = Appliance(id)
$app = Plugwise.Appliances[index]
```

A new appliance object can be created by

```
$app = Plugwise.CreateAppliance(name)
```

and deleted with

```
Plugwise.DeleteAppliance(appliance)
```

	Method	Description	Example	Result
	ClassName	The class name of the object		
	DoNotSwitchOff	True if the appliance is flagged not to switch off.		
	SetDoNotSwitchOff(bool)			
	FirstSeenDate	First moment the module was online after it was attached to the appliance. This is also the start point for logging of the appliance.		
*	SetFirstSeenDate(dateTime)			
*	GetPersistent(name, default)	Retrieve custom info name for this object from database or use default if name is not set.		
*	SetPersistent(name, value)			
	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.		
	IsOnline	True if the (module of the) appliance is online.		
	IsProducer	True if the (module of the) measures energy production.		
	ImageName	Name of the virtual image file		
	LastSeenDate	Timestamp of last contact		
	LastSeenSeconds	Seconds past since last contact		
*	Log(startdate [, enddate [, tarifftype]])	Returns the log data of type tarifftype of the appliance for the specified date or period. tarifftype can be 1 for usage or 257 for production.		
	Module	Module to which the appliance is attached		
	Name	Name of the appliance		
	SetName(string)			
	NotInNetwork	True if the appliance is (temporarily) not part of the network and should be ignored.		
	SetNotInNetwork(bool)			
	PowerState	Power state of the appliance: 'on' or 'off'		
	PowerUsage	Last known power usage		
	RespondToBroadcast	True if the (module of the) appliance responds to switching broadcasts.		
	Runtime	Array of user defined values for this object that exists as long as Source is running.		
	Schedule	Assigned schedule or Null		
*	SetSchedule(schedule)	Assign a schedule or Null to unassign.		
	SkipInTotals	Ignore this appliances when summarizing usage, totals etc for lists of appliances. For instance with Group.TotalUsage.		
	SetSkipInTotals(bool)			

			Example	Result
	StatusImageName	Name of the virtual image that includes the status	`<img src="<%=Plugwise.ImagesPath%>32/<%=.StatusImageName%>.png">`	`<img src="/pwimg/32/appliance_on.png">`
	SwitchOff()	Switch the (module of the) appliance off		
	SwitchOn()	Switch the (module of the) appliance on		
	TotalUsage	Total power usage since the last counter reset. Setting this value by script will **not** reset the `TotalUsageStartDate`		
	SetTotalUsage(*float*)			
	TotalUsageStartDate	Date from witch on TotalUsage is calculated.		
	SetTotalUsageStartDate(*date*)			
	TotalUsageToday	Total power usage for today		
	Type	Appliance type		
	SetType(*string*)			
	TypeText	Appliance type translated to the current language		
	UUId	Universally Unique IDentifier		

Group

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

An existing group object can be obtained in 3 different ways

```
$grp = Group(name)
$grp = Group(id)
$grp = Plugwise.Groups[index]
```

A new group object can be created by

```
$grp = Plugwise.CreateGroup(name)
```

and deleted with

```
Plugwise.DeleteGroup(group)
```

	Method	Description	Example	Result
*	Add(*appliance*)	Adds appliance to the group		
	Appliances	Array of appliances which are member of the group		
	BroadcastMacAddress	The virtual MAC Address of the groups used for broadcasts. Can be empty; no broadcasts used.		
	SetUseBroadcast(*bool*)	Use broadcasts for switching all appliances in the group or not.		
	ClassName	The class name of the object		
*	GetPersistent(*name*, *default*)	Retrieve custom info *name* for this object from database or use *default* if *name* is not set.		
*	SetPersistent(*name, value*)			
	Hidden	`True` If the group is not visible in any screen except the Groups screen.		
	SetHidden(*bool*)			
	Id	Internal ID of the group		
*	Log(*startdate* [, *enddate* [, *tarifftype*]])	Returns the log data of type *tarifftype* of the group's appliances for the specified date or period. *tarifftype* can be 1 for usage or 257 for production.		
	Name	Name of the group		
	SetName(*string*)			
	PowerState	`Off` if all the modules attached to the room's appliances that are online and do not have the NotInNetwork flag are switched off. Otherwise `On`		
	PowerUsage	Total of the group's appliances last known power usage. Note: Appliances with `SkipInTotals` flag set are ignored. Unless all appliances of the group have this flags set, then the flag is ignored.		
*	Remove(*appliance*)	Removes appliance from the group		
	Runtime	Array of user defined values for this object that exists as long as Source is running.		

	Schedule	Assigned schedule or Null		
*	SendSchedules()	For each assigned appliance send its schedules or disable if it has none.		
*	SetSchedule(*schedule*)	Assign a schedule or Null to unassign.		
	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. Note: Appliances with SkipInTotals flag set are ignored. Unless all appliances of the group have this flags set, then the flag is ignored.		
	TotalUsageToday	Total power all the appliances usage for today. Note: Appliances with SkipInTotals flag set are ignored. Unless all appliances of the group have this flags set, then the flag is ignored.		
	Type	Type of the group		
*	SetType(*typename*)	Set the type of the group: possible values are 'report', 'application' and 'switching'. Note: If type is 'report' the Hidden flag is set to False. For other values it is set to True.		
	TypeText	Translated name for type of the group		
	UUId	Universally Unique IDentifier		

Module

The Module object is the representation of the 'Module' or 'Plug' 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. Exceptions are CloseRelay(), OpenRelay() and GetPowerUsage(). They will wait until a valid answer is received or the given timeout has expired.

An existing module object can be obtained in 3 different ways

```
$mod = Module(name)
$mod = Module(macaddress)
$mod = Module(id)
$mod = Plugwise.Modules[index]
```

A new module object can be created by

```
$mod = Plugwise.CreateModule(macaddress)
```

and deleted with

```
Plugwise.DeleteModule(module)
```

	Method	Description	Example	Result
	Appliance	The assigned appliance		
*	Add(*appliance*)	Attaches the module to the appliance. A module can only be attached to 1 module and vice versa.		
	ClassName	Class name of the object		
	CloseRelay(*timeout*, *retries*)	Close the relay; switch on the connected appliance. The result is True if the module did close the relay. Note: The maximum possible 'hang time' for the command is timeout * (retries+1) seconds.	`<%` `$modid=Request.Get["modid"]` `$mod=Plugwise.Modules[$modid]` `if $mod.IsOpen` `  $res= $mod.CloseRelay(4,0)` `else` `  $res= $mod.OpenRelay(4,0)` `/if` `if $res` `  echo "Module switched to: ",$mod.RelayState," "` `else` `  echo "Module switching failed! "` `/if` `%>`	
	FirmwareDate	Timestamp of firmware.		
	FirmwareVersion	Version string of firmware		
	FirstSeenDate	Timestamp of first contact.		

*	`SetFirstSeenDate(`*`dateTime`*`)`			
	FirstSeenLogIndex	Current internal logging index of the module at the time of `FirstSeenDate`		
*	`SetFirstSeenLogIndex(`*`int`*`)`			
*	GetPersistent(*name*, *default*)	Retrieve custom info *name* for this object from database or use *default* if *name* is not set.		
*	`SetPersistent(`*`name, value`*`)`			
*	GetInfo(*timeout*, *retries*)	Requests the node info from the module. The result is `True` if the module did return the node info usage or. The new info is used to update the module's properties.		
	GetPowerUsage(*timeout*, *retries*)	Requests the current measured power usage of the module. The result is `True` if the module did return the usage or if a power usage request is pending, because the relay just closed. The new value is stored in `PowerUsage`.		
	HardwareVersion	Version string of hardware		
	Humidity	Sense: Last reported humidity		
	Id	Internal ID of the module		
	IsClosed	`True` if the relay of module is closed (power is on).		
	IsOnline	`True` if the module is online.		
	IsOpen	`True` if the relay of module is open (power is off).		
	ImageName	Name of the virtual image file		
	LastCompletedLogIndex	Oldest internal logging index of the module of which all data is retrieved and processed.		
*	`SetlastCompletedLogIndex(`*`int`*`)`			
	LastSeenDate	Timestamp of last contact		
	LastSeenSeconds	Seconds past since last contact		
	LastTelegram	If the module is a P0 or P1 reader, this is the last received telegram from the meter		
	MacAddress	MAC address (hardware address) of the module.		
	Name	Name of the module		
	`SetName(`*`string`*`)`			
	Network	Network the module is member off.		
	OpenRelay(*timeout*, *retries*)	Open the relay; switch off the connected appliance. The result is `True` if the module did open the relay.	See `CloseRelay()`	
	PowerUsage	Last known power usage		
	RelayState	Switch state of the relay: 'open' or 'closed'		
*	Remove(*appliance*)	Detaches the appliance from the module.		
	Runtime	Array of user defined values for this object that exists as long as Source is running.		
	Status	Status of the module: 'online', 'offline' of 'unknown'		
	StatusImageName	Name of the virtual image that includes the status	`<img src="<%=Plugwise.ImagesPath%>32/<%=.StatusImageName%>.png">`	`<img src="/pwimg/32/appliance_on.png">`
	Type	Module type		
*	`SetType(`*`type`*`)`			
	Temperature	Sense: Last reported temperature		
	TypeText	Module type translated to the current language		
	UUId	Universally Unique IDentifier		

Network

The Network object is the representation of the 'Network entity in the application. Normally the Network entity is only shown when the application controls more than 1 network.

An existing network object can be obtained in 3 different ways

```
$netw = Network(name)
$netw = Network(macaddress)
$netw = Network(id)
$netw = Plugwise.Networks[index]
```

A new network object can be created by

```
$netw = Plugwise.CreateNetwork(macaddress)
```

and deleted with

```
Plugwise.DeleteNetwork(network)
```

	Method	Description	Example	Result
*	Add(module)	Assigns the module to the network. A module can only be assigned to 1 network.		
	ClassName	Class name of the object		
*	ExpectedOnlineCount	Number of modules that should be online excluding SEDs and those that are flagged 'NotInNetwork'.		
*	GetPersistent(name, default)	Retrieve custom info name for this object from database or use default if name is not set.		
*	SetPersistent(name, value)			
*	GetModuleList()	Returns a list of modules known to the NC. Note: This action blocks the webserver for at least 30 seconds.		
	Id	Internal ID of the module		
	ImageName	Name of the virtual image file		
	MacAddress	MAC address (hardware address) of the module.		
	MC	The Stick module of the network		
	Modules	Array of modules which are assigned to the network		
	Name	Name of the network		
	SetName(string)			
	NC	The Circle+ module of the network		
*	OnlineCount	Number of online modules excluding SEDs and those that are flagged 'NotInNetwork'.		
	PowerUsage	Total of last known power usage of all modules.		
*	Quality	Percentage of online modules excluding SEDs and those that are flagged 'NotInNetwork'.		
*	Remove(appliance)	Detaches the appliance from the module.		
	Runtime	Array of user defined values for this object that exists as long as Source is running.		
	Status	Status of the network: 'online', 'offline'		
	StatusImageName	Name of the virtual image that includes the status	`<img src="<%=Plugwise.ImagesPath%>32/<%=.StatusImageName%>.png">`	`<img src="/pwimg/32/appliance_on.png">`
	SwitchOn()	Switch on the all modules in the network		
	SwitchOff()	Switch off the all modules in the network		
	UUId	Universally Unique Identifier		

Room

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

A new room object can be created with

```
$room = Plugwise.CreateRoom(name)
```

An existing room object can be obtained in 3 different ways

```
$room = Room(name)
$room = Room(id)
$room = Plugwise.Rooms[index]
```

and deleted with

```
Plugwise.DeleteRoom(room)
```

	Method	Description	Example	Result
	Appliances	Array of appliances which are assigned to the room		
*	Add(appliance)	Assigns the appliance to the room		
	ClassName	The class name of the object		
*	GetPersistent(name, default)	Retrieve custom info *name* for this object from database or use *default* if *name* is not set.		
*	SetPersistent(name, value)			
	Id	Internal ID of the room		
*	Log(startdate [, enddate [, tarifftype]])	Returns the log data of type *tarifftype* of the room's appliances for the specified date or period. *tarifftype* can be 1 for usage or 257 for production.		
	Name	Name of the room		
	SetName(string)			
	PowerState	Off if all the modules attached to the room's appliances that are online and do not have the NotInNetwork flag are switched off. Otherwise On		
	PowerUsage	Total of the appliances last known power usage. Note: Appliances with SkipInTotals flag set are ignored. Unless all appliances of the room have this flags set, then the flag is ignored.		
*	Remove(appliance)	Removes appliance from the room		
	Runtime	Array of user defined values for this object that exists as long as Source is running.		
*	SendSchedules()	For each assigned appliance send its schedules or disable if it has none.		
	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. Note: Appliances with SkipInTotals flag set are ignored. Unless all appliances of the room have this flags set, then the flag is ignored.		
	TotalUsageToday	Total power all the appliances usage for today. Note: Appliances with SkipInTotals flag set are ignored. Unless all appliances of the room have this flags set, then the flag is ignored.		
	TemperatureAndHumidity	Array with last reported temperature and humidity or the average if more than one Sense are linked to the room.		
	Type	Room type id		
	SetType(string)	Room type id		
	TypeText	Room type translated to the current language		
	UUId	Universally Unique IDentifier		

Schedule

The Schedule object is the representation of the 'Switching schedule' entity in the application.

A new schedule object can be created with

```
$sched = Plugwise.CreateSchedule(name)
```

An existing schedule object can be obtained in 3 different ways

```
$sched = Schedule(name)
$sched = Schedule (id)
$sched = Plugwise.Schedules[index]
```

and deleted with

```
Plugwise.DeleteSchedule(schedule)
```

	Method	Description	Example	Result
	Appliances	Array of appliances to which the schedule has been assigned directly.		
*	GetPersistent(*name*, *default*)	Retrieve custom info *name* for this object from database or use *default* if *name* is not set.		
*	SetPersistent(*name, value*)			
	AssignedAppliances	Array of appliances to which the schedule has been assigned directly or indirectly via a group or room.		
	Groups	Array of groups to which the schedule has been assigned.		
	Name	Name of the group		
	SetName(*string*)	Name of the group		
	Runtime	Array of user defined values for this object that exists as long as Source is running.		
*	Send()	Sends the schedule to the (modules of) the assigned appliances. The method returns immediately, the sending is done in the background. The result value is the number of modules affected by the new schedule.		
	StatusImageName	Name of the virtual image that includes the status		
	Values	Array of 7 arrays (days) with 96 values (4 quarters * 24 hours). -1 means 'On', 0 means ' Off', any positive value represents the standby value. The keys of the 7 arrays are "mo", "tu", "we", "th", "fr", "sa", "su" and are available via `Plugwise.DayCodes`		
*	SetValues(*array*)			
	UUId	Universally Unique IDentifier		

Tariff

The Tariff object is the representation of the 'Tariff' entity in the application.
A new tariff object can be created with

```
$tar = Plugwise.CreateTariff(name)
```

An existing tariff object can be obtained in 4 different ways

```
$tar = Tariff(name)
$tar = Tariff(id)
$tar = Tariff(Date[,type])
$tar = Plugwise.Tariffs[index]
```

and deleted with

```
Plugwise.DeleteTariff(tariff)
```

	Method	Description	Example	Result
	ClassName	The class name of the object		
	CO2Emission	CO_2 emission in kg per kWh		
*	SetCO2Emission(*float*)			
	CompanyName	Company name property		
	SetCompanyName(*string*)			
	EndDate	End date of the tariff period		
*	SetEndDate(*date*)			
*	GetPersistent(*name, default*)	Retrieve custom info *name* for this object from database or use *default* if *name* is not set.		

			Example	Result
*	`SetPersistent(`*`name, value`*`)`			
	`HasPeakTariff`	Tariff has a split tariff structure		
	`Id`	Internal ID of the tariff		
	`IsPeakTime(`*`date`*`)`	`True` if the peak tariff should be used for the given timestamp. *`date`* must be between with the tariff's start and end date.		
	`IsProducing`	`True` if the tariff is for producing energy (Type >= 256)		
	`Name`	Name of the tariff		
	`SetName(`*`string`*`)`			
	`PeakDays`	Array of 2 letter day of week codes on which the peak tariff should be used.		
*	`SetPeakDays(`*`array`*`)`			
	`PeakEndHour`	Last hour of the daily peak period, 0 to 23		
*	`SetPeakEndHour(`*`hour`*`)`			
	`PeakStartHour`	First hour of the daily peak period, 0 to 23 -1 means no peak period		
*	`SetPeakStartHour(`*`hour`*`)`			
	`PeakTariff`	kWh rate during peak time		
*	`SetPeakTariff(`*`float`*`)`			
	`ProductName`	Product name		
	`Runtime`	Array of user defined values for this object that exists as long as Source is running.		
*	`SetProductName(`*`string`*`)`			
	`StartDate`	Start date of the tariff period		
*	`SetStartDate(`*`date`*`)`			
	`Tariff`	kWh rate for normal or off-peak time		
*	`SetTariff(`*`float`*`)`			
	`Type`	tariff type id		
*	`SetType(`*`int`*`)`			
	`TypeText`	Tariff type translated to the current language		
	`UUId`	Universally Unique Identifier		

Trigger

The Trigger object is the representation of the 'Trigger' entity in the application. It is linked to an event of a module like pressing a Switch button or detecting movement by a Scan. A new trigger object cannot be created via PTE. It is automatically created for the corresponding Switch, Sense or Scan when fired (button pressed, movement detected etc.).

An trigger object can be obtained in 3 different ways

```
$tgr = Trigger(name)
$tgr = Trigger(id)
$tgr = Plugwise.Triggers[index]
```

	Method	Description	Example	Result
	`ClassName`	The class name of the object		
	`EventCount`	Number of times the event of the trigger occurred		
	`SetEventCOunt(`*`integer`*`)`			
	`Group`	The group to which the trigger is linked, if any		
*	`GetPersistent(`*`name, default`*`)`	Retrieve custom info *name* for this object from database or use *default* if *name* is not set.		
*	`SetPersistent(`*`name, value`*`)`			
	`SetGroup(`*`string`*`)`			
	`Id`	Internal ID of the trigger		
	`Module`	The module to which the trigger belongs to		
	`Name`	Name of the trigger		
	`SetName(`*`string`*`)`			
	`Runtime`	Array of user defined values for this object that exists as long as Source is running.		
	`Type`	Trigger type id (same as the module type)		
	`TypeText`	Trigger type translated to the current language		
*	`UUId`	Universally Unique Identifier		

Built-in icons

The built icons in Source can be accessed via the url /pwimg/*size*/*name*.png as transparent PNG images. In a script you can use Plugwise.ImagesPath as the base path. The '*size*' parameter is the width and height of the icon like 20, 32 or 48. All icons are square. The '*name*' can include the status like 'on', 'off' or 'locked'. The `StatusImageName` property of Appliance or Module, contains the full icon name, including the status.

```
<%
foreach Plugwise.Appliances
  echo .Name,': ',.StatusImageName,' <img src="',Plugwise.ImagesPath,'32/',.StatusImageName,'.png"><br>'
/foreach
%>
```

Generating graphs

The same graphs as shown in the Reports screen of Source can be generated via the url /pwgraph/?*parameters*. In a script you can use Plugwise.GraphsPath as the base path. For '*parameters*' see following table. Except 'width' and 'height', all parameters are optional.

Parameter	Purpose	Default
from=*date*	Start date of the period in format YYYY-MM-DD	today
to=*date*	End date of the period in format YYYY-MM-DD	Same as start date
interval=*interval*	Data interval: y = year, m = month, w = week, d = day, h = hour	hour
view=*type*	Type of view: u = usage/production, e = CO_2 emission, c = costs, t = climate, o = occupation, m = On/Off	usage/production
legend=*show*	Show or hide the graph's legend: 1 or on = show, 0 or off = hide	show legend
title=*text*	Title on graph	no title
zoom=*factor*	Resize graph by *factor*. Using a *factor* < 1 gives better image quality than letting the browser resize the image on display.	1, no resizing
appids=*ids*	List of comma separated appliance ids for which to show the graph.	all appliances
grpids=*ids*	List of comma separated group ids for which to show the graph.	all appliances
rmids=*ids*	List of comma separated room ids for which to show the graph.	all appliances
width=*width*	Width of graph in pixels.	mandatory
height=*height*	Height of graph in pixels.	mandatory

A custom color scheme for the graph can be set with the Plugwise.SetColorScheme(*array*) method (see also the Plugwise object in this document). The color scheme is only valid within the same session, so different users can have different color schemes at the same time.

A color represents an ARGB value, this is a 32 bit value where the highest 8 bits define the alpha component (transparency), the following 8 bits the red component, next the green component and then the blue component. For example, 0x00ff0000 represents red, 0xff represents blue and 0x80ffffff is half transparent white.

Name	Purpose	Default
background	Background color	0xffffff (white)
edge	Edge of the graph	0xffffff (white)
border	Border of the image	0xffffff (white)
grid	Grid (horizontal reference lines) in graph	0xd0d0d0 (light grey)
labels	Text labels	0x000000 (black)
usage	Usage representation, also off-peak	0x8d96c8 (blue)
production	Production representation, also off-peak	0x8dc78f (green)
peakusage	Peak usage representation	0xbac9ff (light blue)
peakproduction	Peak production representation	0x9bff97 (light green)
totalusage	Total usage line	0x800000 (red)
totalproduction	Total production line	0x8000 (green)

To prevent unnecessary processing, the webserver uses a simple caching mechanism. Every served graph is saved for 1 minute based on the request string. When a graph is requested the webserver will look for a cached image of less than 1 minute old, that was generated with exactly the same request string and color scheme. If found, the existing image is server, if not, a new graph is generated, saved and served.

Events

Starting from version 2.23 Plugwise Source supports event scripting. This means that you can run a script whenever a certain predefined event occurs.

In Source an event is something that happens to an object like an appliance or module. It is identified and named by the conditions under which it occurs. Almost every object in Source has events defined. For instance, for an appliance the event type 'OnSwitched' happens when an appliance is switched, or to be more precise, it happens when Source detects the appliance is switched.
To make use of events, you need the license feature 'Events'.

> Although the event system uses the same scripting engine, it is not a part of the webserver. You do not need to enable the webserver to make use of event scripting.

Event handlers

An event handler is a function that is called by Source when a certain type of event occurs. For example the OnSwitched() event of an appliance:

```
Function OnSwitched($sender, $switchedOn)
/Function
```

All event types have $sender as the first argument. It is the reference to the object to which the event occurs. In this way you can use the same event handler for different objects if that is convenient. The other arguments are event type specific.

Each object with events can have an OnInit() handler. This handler is called when Source is started and whenever the file that contains the script code is changed. You can use this handler for one time initialization:

```
Function OnInit($sender)
  $sender.Runtime["SwitchedOnCounter"]=0
  $sender.Runtime["SwitchedOffCounter"]=0
/Function

Function OnSwitched($sender, $switchedOn)
  If $switchedOn
    $sender.Runtime["SwitchedOnCounter"]+=1
  Else
    $sender.Runtime["SwitchedOffCounter"]+=1
  /If
/Function
```

In the above example the Runtime[] property is used to count the number of times the appliance is switched on and off. OnInit() is used to initialize the counters to zero.

Appliance
```
OnOnlineStatus($sender)
```
Online status of appliance has changed.

```
OnSwitched($sender, $switchedOn)
```
Appliance is switched on or off.

```
OnPowerZero($sender)
```
Rounded value of the current power usage equals 0.

```
OnPowerUsage($sender, $usage)
```
Current power usage has changed. Source compares the rounded values of latest and the previous measurement.

Group
```
OnBroadcast($sender, $switchedOn, $eventId)
```
Source received a switching broadcast related to the group.

Module (power meter)
```
OnOnlineStatus($sender)
```
Online status of module has changed.

```
OnSwitched($sender, $switchedOn)
```
Module is switched on or off

```
OnPowerZero($sender)
```
Rounded value of the current power usage equals 0.

```
OnPowerUsage($sender, $usage)
```
Current power usage has changed. Source compares the rounded values of latest and the previous measurement.

Module (Switch, Scan)
```
OnBroadcast($sender, $switchedOn, $eventId)
```
Source received a switching broadcast Sent by the module.

Module (Sense)
```
OnBroadcast($sender, $switchedOn, $eventId)
```
Source received a switching broadcast Sent by the module.

```
OnReport($sender)
```
Source received a measuring report from the module.

Schedule
```
OnMinute($sender, $datetime, $value)
```
Occurs every minute. $datetime is the current timestamp, $value is the value as defined in the schedule for the current quarter of the hour.

```
OnEdge($sender, $datetime, $value, $delta)
```
Occurs every 15 minutes, but only if the new quarter has a different value defined in the schedule than the previous quarter. $delta is the difference between these to values.

Switching Trigger
```
OnBroadcast($sender, $switchedOn)
```
Source received a switching broadcast Sent by the module to which the trigger belongs.

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['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 enclosed in quotation marks:

```
<html><body><%
// set to posted values or an empty array
$cks=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>
```

Browser sessions

The engine uses a server side cookie called '_PLUSID_ ' to store the session id of the http client (i.e. browser). If the client does not support cookies, you can create a session by adding a '_PLUSID_' parameter with a (unique) value to the URL:

```
http://server:8080/sessiontest.html?_PLUSID_=12345
```

Syntax highlighting

No editor supports the PTE syntax by default, but most will do a decent job when the syntax is set to PHP. In our experience PSPad (http://www.pspad.com/) handles this very well.
Start PSPad and open the program setting dialog via *Settings → Program Settings*
In left column select *Multihighlighter*
- Check *Enable HTML Multi-highlighter*
- Set *For <%..%> use* to *PHP*
- Under *Open in Multi-highlighter* check *PHP*

Optionally you can make PSPad the default editor for PTE files:
In left column select *Registered File Types*
- Under *Type:* fill in *.pte* and press *Add New*