Designing and integrating user designed web pages for ProfiLux 3

The ProfiLux main page *index.html* provides a link *user web page*, this links to the file *user.html*.
It is easy to integrate this user file in the ProfiLux 3 web server:

1. Create a file user.html
2. Upload this file with TFTP to the ProfiLux 3

It is possible to upload several files, these files may link to each other. ProfiLux 3 offers 256 kB space for user files.

To access ProfiLux' internal data are 2 options given:

Using ProfiLux-specific tokens

ProfiLux scans the requested file when it has a certain file extension (only *.html, *.rss and *.xml are scanned, all other file types are passed through unmodified!) for these tokens and replaces them with live data. The parsing of a file takes some time, if your HTML-file doesn't have any tokes which have to be evaluated then use the file extension *.htm, these files will not be scanned and the output will be faster.

List of tokens

Token	Returns	Min. firmware version
$$SENC[*n*]$$	Complete name, actual value and state of sensor *n*	
$$SENL[*n*]$$	Complete name, actual value and state of sensor *n*. long (incl. description)	5.08
$$SENV[*n*]$$	Actual value of sensor *n* returns "---" if this sensor is disabled	
$$SENN[*n*]$$	Name of sensor *n*	
$$SENS[*n*]$$	Short name of sensor *n*	
$$SEND[*n*]$$	Description of sensor *n*	5.08
$$SENT[*n*]$$	Type of sensor *n* as numerical value 0 = none, 1 = temp., 2 = pH, 3 = redox, 4 = cond. Freshwater, 5 = cond. Saltwater, 6 = none, 7 = humidity, 8 = airtemp., 9 = oxygen, 10 = voltage	5.09
$$SENX[*n*]$$	Nominal value of sensor *n*	5.09
$$SENF[*n*]$$	Format of nominal and actual value of sensor *n* Returns *int*:*dec*:*unit*, where *int* is max. number of digits before the decimalpoint *dec* is the number of digits after the decimalpoint *unit* is the unit for this sensor	5.15
$$SENO[*n*]$$	Operation state of sensor *n*: "+" = control upwards	5.15

© GHL Advanced Technology GmbH & Co. KG

	"-" = control downwards "*" = cooling (only for temp. sensor) "#" = main heater (only for temp. sensor) "~" = bottom heater (only for temp. sensor) "!" = alarm	
$$SENR[*n*]$$	Nominal value range of sensor *n* Return *max…min*, where *min* is the minimal allowed value and *max* is the maximal allowed value both without unit	5.15
$$LEVV[*n*]$$	Value of level sensor *n*	
$$LEVN[*n*]$$	Name of level sensor *n*	
$$ILLC[*n*]$$	Complete name and value of illumination *n*	
$$ILLL[*n*]$$	Complete state of illumination *n* long (incl. description)	5.08
$$ILLV[*n*]$$	Value of illumination *n*	
$$ILLN[*n*]$$	Name of illumination *n*	
$$ILLD[*n*]$$	Description of illumination *n*	5.08
$$ILLP[*n*]$$	Properties of illumination *n* "d" = dimmable "u" = used (has dimming points)	5.15
$$ILCL[*n*]$$	Color of illumination *n*	6.08
$$PROM$$	ProfiLux model name	
$$FWVS$$	Firmware version	
$$FWDA$$	Firmware date	
$$SERI$$	Serial number	
$$SWIS[*n*]$$	State of switch output *n*: "off" = off "on" = on	
$$SWIV[n]$$	State of switch output n: "0" = off "1" = on	
$$SWIN[*n*]$$	Name of switch output *n*	
$$SWID[*n*]$$	Description of switch output *n*	5.09
$$SWIP[*n*]$$	Properties of switch output *n* "u" = used (has other function than *always off*)	5.15
$$SWIM[n]$$	Switch operation mode "1" = manual "0" = auto	6.06
$$DATE$$	Current date in ProfiLux	
$$TIME$$	Current time in ProfiLux	
$$GCOV[*n*]$$	Get code value – returns a string with the value(raw data) of any parameter (code) *n* of the ProfiLux	
$$GWBS[*n*]$$	Get web string – returns a string which can be used for creating multi language pages	
$$ALMS$$	Current alarm state, if there are several alarms active the alarm sources will be displayed alternating with each invoke of this token	
$$ALMS[*n*]$$	Current alarm state, n determines which alarm source should be displayed Since it is unknown if and how many alarms are active	5.05e

© GHL Advanced Technology GmbH & Co. KG

	the use of the indexed alarm-token makes only sense if used in a loop, see example below	
$$AMSE$$	Alarm mask all sensors	5.16
$$AMLC$$	Alarm mask all level controls	5.16
$$AMMI$$	Alarm mask miscellaneous (Bits 11-8: flow sensors, Bit 0: PAB-alarm)	5.16
$$TDRS$$	Current time and date for RSS feed	
$$EXWA$$	External web address of ProfiLux	
$$OTVI[*n*]$$	Actual output voltage of 1-10V-interface *n*	5.04
$$SWIC[*n*]$$	Actual current of switch output *n*	5.05f
$$LVCN[*n*]$$	Name of level control *n*	5.15
$$LVCD[*n*]$$	Description of level control *n*	5.15
$$LVCO[*n*]$$	Operation state of level control *n*: “+” = fill water “-“ = drain water “!” = alarm “1” = first level sensor of this control indicates contact “2” = second level sensor of this control indicates contact (if present)	5.15
$$LANG$$	Returns the language ProfiLux is using, e.g.: “Deutsch” “English” “Espanol” “Francais”	
$$FPSN[*n*]$$	Name of feedpause *n*	5.15
$$FPSD[*n*]$$	Description of feedpause *n*	5.15
$$FPSR[*n*]$$	Remaining time for feedpause *n* “0 s” means this feedpause is not active	5.15
$$MAIN[*n*]$$	Name of maintenance *n*	5.15
$$MAID[*n*]$$	Description of maintenance *n*	5.15
$$MAIR[*n*]$$	Remaining time for maintenance *n* “0 s” means this maintenance is not active	5.15
$$AWCN[*n*]$$	Name of automatic waterchange *n*	5.15
$$AWCS[*n*]$$	State of automatic waterchange *n*: “+” = fill water “-“ = drain water “!” = alarm otherwise idle	5.15
$$THSS$$	State of thunderstorm: “0” = no thunderstorm “1” = thunderstorm in progress	5.15
$$MEAS[*n*]$$	Measurement value sample *n* This returns a sample of a certain sensor, the index of the sensor is determined by the suffix of the loaded file, e.g. sensordata003.txt returns values of sensor 3. Index n = 0 returns the newest sample. The format is: *yyyy-mm-dd hh:mm value*, where *yyyy-mm-dd* is the date (year, month, day) *hh:mm* is the time (hours, minutes)	5.15

© GHL Advanced Technology GmbH & Co. KG

	value is the measured value with unit, e.g.: 2012-12-10 16:07 21.300 C This token is usually used in a loop, see example below. In order to retrieve only samples until a certain time and date the code	
$$EHMF$$	Actual flow of EHEIM filter in litres/hour, "---" if no EHEIM filter is present	5.16
$$EHMS$$	Time to next service for EHEIM filter in hours, "---" if no EHEIM filter is present	5.16
$$FLSV[*n*]$$	Actual value of flow sensor *n* returns "---" if this sensor is disabled	5.17
$$FLSN[*n*]$$	Name of flow sensor *n*	5.17
$$FLSF[*n*]$$	Format of nominal and actual value of sensor *n* Returns *int*:*dec*:*unit*, where *int* is max. number of digits before the decimalpoint *dec* is the number of digits after the decimalpoint *unit* is the unit for this sensor	5.17
$$FLSD[*n*]$$	Description of flow sensor *n*	5.17
$$RMDT$$	Text of first active reminder	5.17
$$RMDT[*n*]$$	Text of reminder n, if it is active	5.17
$$RMDP[n]$$	Reminder properties "m" = multiple reminder	6.06
$$RMDD[n]$$	Days count until reminder *n* is shown	6.06
$$DOSN[*n*]$$	Name of doser *n*	6.08
$$DOSL[*n*]$$	Last for days doser *n*	6.08
$$DOSF[*n*]$$	Filling level of doser *n*	6.08
$$DOST[*n*]$$	Alarm threshold of doser *n*	6.08
$$DOSS[*n*]$$	State of doser *n*	6.08
$$DOSC[*n*]$$	Capacity of doser *n*	6.08
$$DOSD[*n*]$$	Description of doser *n*	6.08

Some tokens are indexed; in this case you have to replace *n* with the number of the resource you are going to access. Indexes start at 0.

If you write "*x*" for the index then this index while be replaced with the current loop counter, see tokens for creating a loop below.

Examples:

HTML-code	Result
My sensor: $$SENN[0]$$	My sensor: pH 1 (assuming the first sensor is a pH-sensor)
My ProfiLux is a $$PROM$$	My ProfiLux is a ProfiLux 3 eX (if it is an eX-version)
Firmware: $$GCOV[0]$$	Firmware: 500 (Code 0 returns the firmware version)

Furthermore there are tokens for creating a loop:

Token	Meaning
$$REPS[*c*][*o*]$$	*Repeat Start*, *c* = count of repeats, *o* = option, following code until $$REPE$$ will be repeated *c* times [*o*] is optional
$$REPE$$	*Repeat End*
$$LCTR$$	Returns current value of the internal counter during executing the loops.

The HTML-code between these 2 tokens will be repeated c times, an internal counter will be counted from 0 to *c* -1. This internal counter can be accessed with the "index" *x*.

The option *o* determines if the loop shall be executed depending on the presence or activation of a resource (= sensor, illumination channel, switch output, …). If [o] is not mentioned than all loops will be executed, if [*o*] is mentioned than *o* can have these values:

Value for the option *o*	Will check presence or activity of this resource:
0	Sensor
1	Illumination
2	Switch output
3	Level-Sensor
4	1-10V-Interface
5	Alarm (ProfiLux firmware 5.05e or higher required)
6	Level control
7	Feedpause
8	Maintenance
9	Waterchange
10	Flow-Sensor
11	Reminder

Hiding text depending on the current login status:

Token	Meaning
$$IFLS[*n*]$$	*If Logged in Start,* the following text until token *$$IFLE$$* will only be shown if the login privileges are at least *n:* 1: Guest or Admin must be logged in 2: Admin must be logged in
$$IFLE$$	*If Logged in End*

Examples:

HTML-code	Result
$$REPS[3]$$ My sensor: $$SENN[x]$$ $$REPE$$	My sensor: pH 1 My sensor: Temp 1 My sensor: Redx 1 (assuming these sensor input were present)
$$REPS[32][1]$$ $$ILLN[x]$$ … $$ILLV[x]$$ $$REPE$$	Illumina. 1 … 100% Illumina. 2 … 79% Illumina. 3 … 0%

© GHL Advanced Technology GmbH & Co. KG

	(if only illumination channels 1-3 are active)

$$REPS[10][5]$$
 $$ALMS[x]$$
$$REPE$$
	list all alarms, e.g.:
	`Alarm:  pH 2`
	`Alarm:  Temp 2`
	`Alarm:  Le.F 1`

In the second example the loop runs from 0 to 31 (32 times), but a loop only produces an output if the resource (here: illumination) is activated.

© GHL Advanced Technology GmbH & Co. KG