CUSTOM SCRIPT PROGRAMMING

The documentation describes how a customized object can be programmed
This control is for developers only, therefore no support is provided.

CONTENT

DESCRIPTION OF TIME FUNCTIONS

- sleep
- sleeps
- getcurrenttime
- getyear
- getmonth
- getday
- gethour
- getminute
- getsecond
- gettimeval
- convertutc2local
- convertlocal2utc

DESCRIPTION OF FILE FUNCTIONS

- fseek

DESCRIPTION OF HTTP FUNCTIONS

- httpget

DESCRIPTION OF STREAM FUNCTIONS

- stream_create
- stream_printf
- stream_write
- stream_flush
- stream_read
- stream_readline
- stream_close

EXAMPLES

- Computing
- Website download per tcp stream
- Connecting to an APC USV, if apcupsd is used on the PC
- Read from RS232 Extension via RS232 stream

GENERAL INFORMATION

UNDERLYING

The programming language PicoC (http://code.google.com/p/picoc/) is used. PicoC is a very small C interpreter for scripting. It was originally written for scripting a UAV's on-board flight system and it is also very suitable for other robotic, embedded and non-embedded applications too.

PicoC Copyright

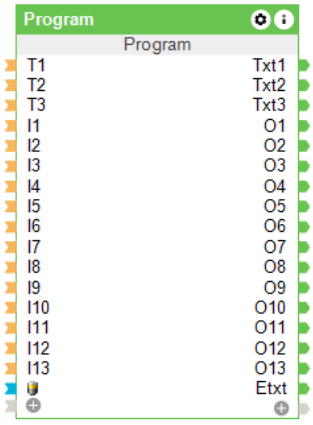
IMPORTANT

- Note that the complete **C specification can not be met!**
- The **program code is interpreted**. Therefore keep the number of lines as **low as possible** and avoid complex program structures.
- Each program runs in a **separate task, asynchronous to the SPS**.
- Outputs with **printf will be written to the log window** of Loxone Config.
- Note also that with a custom program script, the system could crash, so please program your instructions carefully. In case of
- emergency a backup SD card must be on hand, so the system can be made operational again in the case of a custom script causing a problem.
- If you want to run the program indefinitely, add a sleep function implicity into the program loops so that the program does not consume unnecessary CPU time.
- An **integer has 32 bits**, little-endian.
- **128kB are available for heap and stack**. Use memory frugally; do not forget to release memory.
- All functions which return a pointer must **reallocate the memory with free ()**, unless the pointer was passed as a parameter.
- **Time values** are indicated in seconds since 1.1.2009 0 UTC clock.
- **Strings** are encoded in UTF-8.
- **Use mathematical functions sparsely** – the processor in the Miniserver has no mathematical hardware unit.

> Erroneous Program blocks (PicoC) causing the Miniserver to perform safety reboots or other problems will be deactivated by the Miniserver. Those blocks remain deactivated until fixed within Loxone Config.

SPS OBJECTS

There are program objects with 1, 4, 8 and 16 inputs and outputs. You can insert a program object by choosing it under General in the tab Program.
Each program also has an error output Etxt, where interpreter error messages are released. This allows the detection of syntax errors.
The program code can be edited by double-clicking on the program object.



AVAILABLE FUNCTIONS

In the following chapters all available functions are listed and all Miniserver specific functions are described. For all standard C functions, plenty of information can be found in appropriate literature (e.g. www.cplusplus.com/reference/clibrary/).

C LIBRARY FUNCTIONS

```
char *getprogramname();
void printf(char *format, …);
char *sprintf(char *ptr, char *format, …);
void exit();
void *malloc(int size);
void *calloc(int num,int size);
void *realloc(void *ptr,int size);
void free(void *ptr);
int atoi(char *str);
float atof(char *str);
void strcpy(char *dst,char *src);
void strncpy(char *dst,char *src,int len);
int strcmp(char *str1,char *str2);
int strncmp(char *str1,char *str2,int len);
void strcat(char *dst,char *src);
char *strdup(char *str);
char* strstr(char *str,char *strfind);
int strfind(char *str,char *strfind,int pos);
int strlen(char *str);
void memset(void *ptr,int val,int len);
void memcpy(void *dst,void *src,int len);
int memcmp(void *ptr1,void *ptr2,int len);
void errorprintf(char *format,…);
```

Print to standard error device. Only used in shell command.

MATHEMATICAL FUNCTIONS

float sin(float x);
Returns the sine of an angle of x radians.
float cos(float x);
Returns the cosine of an angle of x radians.
float tan(float x);
Returns the tangent of an angle of x radians.
float asin(float x);
Returns the principal value of the arc sine of x, expressed in radians. In trigonometrics, arc sine is the inverse operation of sine.
float acos(float x);
Returns the principal value of the arc cosine of x, expressed in radians. In trigonometrics, arc cosine is the inverse operation of cosine.
float atan(float x);
Returns the principal value of the arc tangent of x, expressed in radians. In trigonometrics, arc tangent is the inverse operation of tangent.
float sinh(float x);
Returns the hyperbolic sine of x.
float cosh(float x);
Returns the hyperbolic cosine of x.
float tanh(float x);
Returns the hyperbolic tangent of x.
float exp(float x);
Returns the base-e exponential function of x, which is the e number raised to the power x.
float fabs(float x);
Returns the absolute value of x
float log(float x);
Returns the natural logarithm of x.
float log10(float x);
Returns the common (base-10) logarithm of x.
float pow(float base,float exponent);
Returns base raised to the power exponent.
float sqrt(float x);
Returns the square root of x.
float round(float x);
Returns rounded value of x.
float ceil(float x);
Returns the smallest integral value that is not less than x.
float floor(float x);
Returns the largest integral value that is not greater than x.

EXTENDED C LIBRARY FUNCTIONS

int batoi(char *str);
float batof(char *str);
char* strstrskip(char *str,char *strfind);
char *index(char *,int);
int lineno();

SPS FUNCTIONS

Following Functions are available:

```
void setlogtext(char *str);
float getio(char *str);
int setio(char *str,float value);
int getinputevent();
float getinput(int input);
char *getinputtext(int input);
void setoutput(int output,float value);
void setoutputtext(int output,char *str);
```

FILE FUNCTIONS

```
FILE *fopen(char *filename,char *mode);
int fclose(FILE *f);
int fprintf(FILE *f,char *format, … );
int fputc(int character,FILE *f);
int fputs(char *str, FILE *f);
int fflush(FILE *f);
int fwrite (void *ptr,int size,int count,FILE *f);
int fgetc(FILE *f);
char *fgets(char *str,int num,FILE *f);
int fread(void *,int size,int count,FILE *f);
int fseek(FILE *f,long offset,int origin);
int remove(char *filename);
int rename(char *oldname,char *newname);
FILE* tmpfile();
char *tmpnam(char* str);
```

HTTP FUNCTIONS

```
char*httpget(char*address,char*page);
```

STREAM-FUNCTIONS

```
STREAM *stream_create(char* filename,int read,int append);
void stream_printf(STREAM* stream,char *format, …);
int stream_write(STREAM* stream,void* ptr,int size);
void stream_flush(STREAM* stream);
int stream_read(STREAM* stream,void* ptr,int size,int timeout);
int stream_readline(STREAM* stream,void* ptr,int maxsize,int timeout);
void stream_close(STREAM* stream);
```

DESCRIPTION OF EXTENDED C LIBRARY FUNCTIONS

atoi

```
int batoi(char *str);
```

Works like the atoi standard C function with the exception that blanks at the beginning of parameter str are allowed.

Example:

```
atoi(" 21"); // return value = NULL
batoi(" 21"); // return value = 21
```

batof

```
float batof(char *str);
```

Works like the atof standard C function with the exception that blanks at the beginning of parameter str are allowed.

Example:

```
atof(" 2.1"); // return value = NULL
batof(" 2.1"); // return value = 2.1
```

strstrskip

```
char* strstrskip(char *str,char *strfind);
```

Works like the strstr standard C function with the exception that the length of parameter strfind is added to the result pointer.

Example:

```
strstr("das ist ein test","ein "); // return value = "ein test"
strstrskip("das ist ein test","ein "); // return value = "test"
```

Index

```
char *index(char *str,int charfind);
```

Returns a pointer to the first occurrence of character charfind in str, or a null pointer if charfind is not part of str.

Example:

```
index("0123456789",0x32);  // return value = "23456789"
```

lineno

```
int lineno();]
```

Returns the current line number in script.

getxmlvalue

```
char *getxmlvalue(char *str,int index,char* name);
```

Returns the value of name in str, or a null pointer if name is not part of str.

Parameters:

str: XML-String to parse
index: the index of occurence of name (first element is 0)
name:
XML-Attribute name to get
The memory where the returned pointer points to must be deallocated with function free().

Example:

```
getxmlvalue("",0,"name");  // return value = "test"
```

getcpuinfo

```
int getcpuinfo();
```

int getcpuinfo();

getheapusage

```
int getheapusage();
```

Returns actual value of system heap in kB.

getmaxheap

```
int getmaxheap();
```

Returns maximum value of system heap in kB.

getspsstatus

```
int getspsstatus();
```

Returns actual number of cycles processed by SPS.

localwebservice

```
char *localwebservice(char *str);
```

Execute webservice and return a pointer to the answer XML string.

Parameters:

str: webservice
The memory where the returned pointer points to must be deallocated with function free().

Example:

```
char* p = localwebservice("dev/sps/status");
free(p);
```

DESCRIPTION OF SPS FUNCTIONS

setlogtext

```
void setlogtext(char *str);
```

Writes the string contained in paramater str to the log window of Loxone Config.

getio

```
float getio(char *str);
```

Returns the value of the speciefied virtual input or a virtual output.

Example:

```
getio("VI1"); // Returns the value of the virtual input VI1
```

setio

```
int setio(char *str,float value);
```

Sets the value of the specified virtual input or a virtual output.

Example:

```
setio("VI1",6.33); // Sets the value of the virtual input VI1 to 6.33
```

getinputevent

```
int getinputevent();
```

Returns a bitmask which contains the changes of inputs (bit 0 = first input of object, starts with text inputs followed by analog inputs).

A return value of 0x02 for example means that input 2 has changed. (see example 1).

getinput

```
float getinput(int input);
```

Returns the value of the analog input specified in parameter input. (0 = first analog input) (see example 1).

getinputtext

```
char *getinputtext(int input);
```

Returns the string of the text input specified in parameter input. (0 = first text input).

setoutput

```
void setoutput(int output,float value);
```

Sets the specified analog output to the specified value. (0 = first analog output) (see example 1).

setoutputtext

```
void setoutputtext(int output,char *str);
```

Sets the spcified text output to the string specified in parameter str. (0 = first text output) (see example 1).

DESCRIPTION OF TIME FUNCTIONS

sleep

```
void sleep(int ms);
```

Sleep in milliseconds.

sleeps

```
void sleeps(int s);
```

getcurrenttime

```
unsigned int getcurrenttime();
```

Get UTC time in seconds since 1.1. 2009.

getyear

```
int getyear(unsigned int time,int local);
```

Get year of specified time. You can set the following parameters:

time:
UTC time in seconds since 2009

local:
Specifies time format: local time when 1, UTC when 0.

getmonth

```
int getmonth(unsigned int time,int local);
```

Get month of specified time. You can set the following parameters:

time: UTC time in seconds since 2009

local: Specifies time format: local time when 1, UTC when 0

getday

```
int getday(unsigned int time,int local);
```

Get day of specified time. You can set the following parameters:

time: UTC time in seconds since 2009
local: Specifies time format: local time when 1, UTC when 0

gethour

```
int gethour(unsigned int time,int local);
```

Get hour of specified time. You can set the following parameters:

time: UTC time in seconds since 2009
local: Specifies time format: local when 1, UTC when 0

getminute

> int getminute(unsigned int time,int local);

Get minute of specified time. You can set the following parameters:

time: UTC time in seconds since 2009
local: Specifies time format: local when 1, UTC when 0

getsecond

> int getsecond(unsigned int time,int local)

Get second of specified time. You can set the following parameters:

time: UTC time in seconds since 2009
local: Specifies time format: local when 1, UTC when 0

gettimeval

> unsigned int gettimeval(int year,int month,int day,int hour,int minutes,int seconds,int local);

Get UTC time in seconds since 2009. If parameters year, month, day, hour, minutes and seconds are UTC time, parameter local must be set to 0. If parameters year, month, day, hour, minutes and seconds are local time, parameter local must be set to 1.

convertutc2local

> unsigned int convertutc2local(unsigned int timeutc);

Converts UTC time to localtime.

convertlocal2utc

> unsigned int convertlocal2utc(unsigned int timelocal);

Converts local time to UTC time.

DESCRIPTION OF FILE FUNCTIONS

fseek

> int fseek(FILE *f,long offset,int origin);

There is a difference to the standard C fseek function: origin must be 0, 1 or 2.
If somebody wants to use the standard C syntax they need to make the following definitions in PicoC script:

```
#define SEEK_CUR    0    // seek from current position
#define SEEK_SET    1    // seek from beginning of file
#define SEEK_END    2    // seek from end of file
```

For all other standard C file functions plenty of information can be found in appropriate literature (e.g.www.cplusplus.com/reference/clibrary/).

DESCRIPTION OF HTTP FUNCTIONS

httpget

```
char *httpget(char *address,char *page);
```

Downloads the specified page from the specified address. You can set the following parameters:

address: Server address e.g. "de.wikipedia.org"
page: Page on the specified server e.g."/wiki/Intelligentes_Wohnen"

The memory where the returned pointer points to must be deallocated with function free().

DESCRIPTION OF STREAM FUNCTIONS

stream_create

```
STREAM *stream_create(char* filename,int read,int append);
```

Creates a file stream, a tcp stream, an udp stream a syslog stream or a RS232/RS485 stream (see example 2 and example 3)

filename:

Value	Description
/path/filename	File-Stream
/dev/tcp/server-address/port	TCP-Stream
/dev/udp/server-address/port	UDP-Stream
/dev/syslog	Syslog-Stream
/dev/tty/name of RS232 or RS485	RS232/RS485 Stream (only)
Extension	stream_read is possible, no stream_write

read: If stream is a file stream: 1 = read file; 0 = write file
append: 1 = append text to file; 0 = overwrite text in file

stream_printf

```
void stream_printf(STREAM* stream,char *format, …);
```

Writes to the output buffer of the specified stream a sequence of data formatted as the format argument specifies. After the format parameter, the function expects at least as many additional arguments as specified in format.

stream_write

```
int stream_write(STREAM* stream,void* ptr,int size);
```

Writes an array of size elements, from the block of memory pointed by ptr to the to the output buffer of the specified stream (see example 2).

The postion indicator of the stream is advanced by the total number of bytes written.
The total number of elements successfully written is returned. If this number differs from the size parameter, it indicates an error.

stream_flush

```
void stream_flush(STREAM* stream);
```

If the given stream is a file stream and the file was opened for writing any unwritten data in the output buffer is written to the file (see example 2).

If the given stream is a tcp stream, an udp stream or a syslog stream any data in the output buffer is sent to the server.

stream_read

```
int stream_read(STREAM* stream,void* ptr,int size,int timeout);
```

Reads an array of size elements from the input buffer of the specified stream and stores them in the block of memory specified by ptr (see example 2).

The timeout is given in milliseconds. The postion indicator of the stream is advanced by the total amount of bytes read. The total number of elements successfully read is returned.

stream_readline

```
int stream_readline(STREAM* stream,void* ptr,int maxsize,int timeout);
```

If the given stream is a file stream the next line of the file is read.

The maximal number of elements to read is specified with parameter maxsize. The timeout is given in milliseconds.

stream_close

```
void stream_close(STREAM* stream)
```

Closes the specified stream and deallocates all memory used by the stream.

EXAMPLES

The examples should show how script programs can be created.

EXAMPLE 1: COMPUTING

This example shows how an own computing is done:
If a change on any input occurs, the value of input 1 multiplied with the value on input 2 plus the value on input 3 is given to output 1.
The formula is output as text on text output 1.

```
// write program here in PicoC
char szBuffer[128];
float f1,f2,f3,f4;
int nEvents;
while(TRUE)
{
nEvents = getinputevent();
if (nEvents & 0xe)
{
f1 = getinput(0);
f2 = getinput(1);
f3 = getinput(2);
f4 = f1 * f2 + f3;
setoutput(0,f4);
sprintf(szBuffer,"/ * %f + %f = %f",f1,f2,f3,f4);
setoutputtext(0,szBuffer);
printf(szBuffer);
}
sleep(100);
}
```

EXAMPLE 2: WEBSITE DOWNLOAD PER TCP STREAM

This example shows how to download a website with a tcp stream and how to save the website in a file.

```
#define BUFF_SIZE 40000
#define RD_BLOCK_SIZE 128
STREAM* pTcpStream = stream_create("/dev/tcp/www.w3.org/80",0,0); // create tcp stream
char* pTcpCmd = "GET / HTTP/1.1\r\nHost: www.w3.org\r\nUser-Agent: LoxLIVE
[en]\r\nContent-Type: text/html; charset=utf-8\r\n\r\n";
stream_write(pTcpStream,pTcpCmd,strlen(pTcpCmd));  // write to output buffer
stream_flush(pTcpStream);                          // flush output buffer
char szBuffer[BUFF_SIZE];
char szTmpBuffer[RD_BLOCK_SIZE];
int nCnt;
int nBytesReceived = 0;
int i = 0;
// read stream
do
{
nCnt = stream_read(pTcpStream,szTmpBuffer,RD_BLOCK_SIZE,4000);
if(nCnt > 0)
strncpy((char*)szBuffer+i*RD_BLOCK_SIZE,szTmpBuffer,nCnt);
nBytesReceived += nCnt;
i++;
}
while (nCnt > 0);
FILE* pw = fopen("/w3.html","w");          // open file
fprintf(pw,"%s",szBuffer);             // write file
printf("Bytes received: %d",nBytesReceived);
fclose(pw);
stream_close(pTcpStream);
```

EXAMPLE 3: CONNECTING TO AN APC USV, IF APCUPSD IS USED ON THE PC

This example shows the actual load (O1), battery capacity (O2) as well as power failures (O3).

```
char* p,*pS;
char szBuffer[1500];
int nLen;
int bOnline = 0;
double dCharge,dLoad;
printf("Start APC watch");
while(TRUE)
{
STREAM* stream = stream_create("/dev/tcp/192.168.1.9/3551",0,0);
if (stream != NULL)
{
szBuffer[0] = 0;
szBuffer[1] = 6;
stream_write(stream,szBuffer,2);
stream_flush(stream);
```

```
stream_write(stream,"status",6);
stream_flush(stream);
nLen = stream_read(stream,szBuffer,2,1000);
nLen = stream_read(stream,szBuffer,sizeof(szBuffer) – 1,1000);
stream_close(stream);
szBuffer[nLen] = 0;
szBuffer[nLen + 1] = 0;
//printf("APC Len=%d",nLen);
pS = szBuffer;
while(*pS)
{
p = strstr(pS,"STATUS   :");
if (p != NULL)
{
p = strstr(pS,"STATUS   : ONLINE");
if (p != NULL)
bOnline = 1;
else
bOnline = 0;
}
else
{
p = strstr(pS,"LOADPCT  :");
if (p != NULL)
{
dLoad = batof(p + 11);
setoutput(0,dLoad);
}
else
{
p = strstr(pS,"BCHARGE  :");
if (p != NULL)
{
dCharge = batof(p + 11);
setoutput(1,dCharge);
break;
}
}
}
pS += (strlen(pS) + 1);
}
if (bOnline)
{
setoutput(2,1);
//printf("Power OK Charge: %f%% Load: %f%%",dCharge,dLoad);
}
else
{
setoutput(2,0);
//printf("Power lost Charge: %f%% Load: %f%%",dCharge,dLoad);
```

```
}
}
else
printf("APC no connection");
sleeps(10);   // wait 10 seconds
}
```

EXAMPLE 4: READ FROM RS232 EXTENSION VIA RS232 STREAM

This example reads the bytes received from RS232 extension and puts it on text output 1. The total number of received bytes is put to the first analog output.

```
#define BUFF_SIZE 256
STREAM* pRs232Stream = stream_create("/dev/tty/my_rs232_extension",0,0); //create rs232 stream
char szBuffer[BUFF_SIZE];
int nCnt;
int nBytesReceived = 0;
while(1)
{
nCnt = stream_read(pRs232Stream,szBuffer,BUFF_SIZE,100); // read stream
nBytesReceived += nCnt;
setoutput(0,(float)nBytesReceived);
setoutputtext(0,szBuffer);
sleep(100);
}
```