Programming Manual

Lua Firmware API

Version: STB00011PE17

Table of Contents

1

Introduction

This document defines the extension to the Lua 5.1.5 API for SATO printers running Lua. It is applicable to all SATO printers unless a printer specific Lua API specification override.

This version (PD1 and later) is applicable for:

- CLxNX firmware later than 1.11.x.
- FX3 firmware later than 5.1.x
- CT4-LX firmware later than 6.0.0.
- TH2 firmware later than 40.00.03.02

2

Abbreviations & Definitions

Lua	Interpreting language. More information can be found at www.lua.org, version 5.1.5
TBD	To be defined / to be decided
API	Application Programming Interface

3

Introduction to Lua

Lua is a interpreted high level language that mixes syntax from many different languages. There's a good reference manual for Lua5.1 at www.lua.org. This section just describes common cases. There are more to each of the listed items below than described here.

3.1.1 Variable names

Variable names start with _ or a-z/A-Z and zero or more letters or digits.

The language is case-sensitive: e.g. Value is a different variable than value.

3.1.2 Common variable types

Common variable types in Lua: boolean, function, nil, number, string, table, userdata and ...

```
-- one line comments start with dash dash and end with newline
```
`boolean` is either `true` or `false`
`function` is a Lua function that can be called
`nil` is the default value of a variable. If assigning `nil` to a variable it is deleted.
`number` are always double type numbers that carry an integral part and a fractional part. The decimal point is always '.', e.g. 15.0
`string` is 0 or more bytes consisting of bytes between 0-255, commonly letters and digits
`table` is the array type/dictionary type for more complex variables.
`userdata` is an extension which SATO uses a lot for our barcodes.
`...` is the magic name for the rest of the arguments, 0 to n arguments.

To determined the type of a variable v, `type(v)` is used, but for `userdata` it is often more useful to display the result from `tostring(v)`.

3.1.3 Variable scope

Variable scope is the term used to define from where the variable is visible. The default scope is the global scope, but if the variable is introduced with local its scope is local in the current code block and sub-blocks.

3.1.4 The first index postion is 1

In Lua the first index position is 1, not 0 as is common in other languages. This applies to strings and to tables.

Examples

```
> str="abc"
> print(str:sub(3,3)) -- prints "c" as 3 is the third letter in str
"c"
> t={2,4,6} -- make a table
> print(t[3]) -- prints the number 6, which is at the third position in t
6
> print(t[0]) -- there is no assigned variable at index 0
nil
```

3.1.5 Assignments

Assignments take this format

```
variable=expression
```

Examples:

```
a=true
f=function(a,b)
  -- this is a code block and the variables a and b are inside the scope
  return a*b
end
n=1138
-- strings start and stop with ' or " depending on the start character
s='hello "world"'
s="hello 'world'"
t={a=false,n=25.4}
t[' space ']="table keys can contain space"
t[false]="table indexes can be anything but nil"
t[f]={5,3} -- table indexes can thus be functions
```

There are more types to read about at www.lua.org

3.1.6 The tricky type coercion

Lua automatically converts numbers to strings and vice versa depending on the expected data type. The conversion is not always what you expect, and perhaps due to that it is dubbed coercion. This is illustrated below:

```
>return "5"*3
15
> return "5"..3
53
> return "5"==5
false
```

Please understand this when comparing values; also learn how to convert from string to number and vice versa. (`tostring(x), tonumber(x[,base]`).

3.1.7 if-then-else-conditionals

```
if expression then
  -- this is a code block
   -- when expression is not nil and not false
elseif another_expression then
  -- this is a code block
  -- elseif is optional
else
  -- this is a code block
  -- else is also optional
end -- end is required
```

In Lua there are two values that are interpreted as false: `nil` and `false`. This is an important distinction to understand.

3.1.8 Loop constructs

Some common loop constructs

```
-- Going through a Lua table
for i,v in pairs(t) do
  -- this is a code block
  print("index:"..tostring(i),"value:"..tostring(v))
end

-- Going through the array-part part of a Lua-table
for i,v in ipairs(t) do
  -- this is a code block
  print("index:"..tostring(i),"value:"..tostring(v))
end

for i=1,#t do
  -- this is a code block
  print("index:"..tostring(i),"value:"..tostring(t[i]))
end

while expression do
  -- this is a code block
  -- run this code while expression is true
end

repeat
  -- this starts the code block
  -- run this code until expression is true
until expression -- and it ends at the expression
```

3.1.9 Comparisons (aka Relational Operators)

```
== -- equal to
~= -- not equal to
<  -- less than
> -- larger than
<=  -- less than or equal to
>= -- larger than or equal to
```

The result from those comparisons are always `true` or `false`.

3.1.10 Logical operators (and,or)

```
sample1=expression1 and expression2
sample2=expression2 or expression3
```

Assigning a default value of "5" can thus be done like this:

```
value=value or "5"
```

3.1.11 Tricks on return values

Lua functions can return multiple return values.

```
function example() return 1,2,3,5 end
```

The example function above returns the four first fibbonacci numbers, and can be used like this:

```
-- assign all 4
local a,b,c,d = example()
-- assign 2 drop 2
local a,b = example()
-- assign 1 and "b" (automatic truncation)
local a,b = example(),"b"
-- pass on just the first value from example() (explicit truncation)
function one() return (example()) end
-- put all return values into table
local t={example()}
```

3.1.12 Object oriented style

For some types Lua allows the object oriented style. It is recognized as `object:method(par1)` and means call the method on object with parameter par1. It works on variable strings, io and on userdata. In Lua it is described as syntactical sugar as the object notation is rewritten internally and automatically to `object.method(object,par1)`.

```
str="this is a string"
```

`str:byte(1)` is equivalent to `string.byte(str,1)`

3.1.13 Object oriented style, metatables

The object oriented style is made possible by the Lua concept of metatable. It is the technique used for userdata to run functions/methods on userdata. For investigation purposes it's good to know how to find out more about them.

Example:

```
> tt=textTTObject.new() for k,v in pairs(getmetatable(tt)) do print(k,v) end
face       function: 0x38f238
codepage   function: 0x38f170
anchor     function: 0x38f2a0
shear      function: 0x38f110
__index    table: 0x38f038
size       function: 0x38f150
dir        function: 0x38f0c8
__tostring function: 0x38f2e0
text       function: 0x38f0f0
__gc       function: 0x38f2c0
__newindex function: 0x38f278
pen        function: 0x38f1f8
font       function: 0x38f088
lineSpacing           function: 0x38f198
clone      function: 0x38f218
fieldSize  function: 0x38f060
info       function: 0x38f1d8
fit        function: 0x38f1b8
pos        function: 0x38f0a8
```

From this you can see there's a method `pos` that can be executed on `tt`. The method `pos` is documented later in this document, and from there the interpretation is that `tt:pos()` returns the hPos and vPos for the object `tt`.

3.1.14 Standard Lua functions

There are many standard Lua functions that are available in the printer, all of them are described on internet, but here's a quick list of the common ones (`t` --table, `f` --function, `e` --expression, `v` --variable, `s` --string, `i` --index, `j` --index)

`ipairs(t), pairs(t)` -- functions to iterate/traverse a table

`b,...= pcall(f,arg1,...)` -- protected call of function f for runtime error handling

`print(...)` -- convert all ... to string values and display (on console)

`tonumber(e[,base]), tostring(e)` -- convert expression to number/string

`type(v)` -- return the Lua type of variable v

`code=require(module-path)` -- load the code from module-path

3.1.15 String functions

Some of the more useful string functions

`a[,b,...]=string.byte(s,[i,j]])` -- the byte value(s) of s

`s=string.char(...)` -- convert all parameters in ... to the corresponding byte character codes.

`s,e=string.find(s, pattern [, init [, plain]])` -- find the start and end position of the pattern in s. The init parameter is the position to start at, the plain parameter can be used to turn of pattern matching, and instead use plain match.
`s=string.format(formatstring,...)` -- format the parameters in ... according to the formatstring
`f=string.gmatch(s,pattern)` -- returns an iterator function that matches pattern in s
`s[,n]=string.gsub(s,pattern, replacement [, n])` -- replaces n or all occurrences of pattern in s with replacement.
`n=string.len (s) or #s` -- return the number of bytes in s.
`s=string.sub(s,i[,j])` -- return the substring between i and j from s

The pattern functions use Lua's regular expression style. This is very useful but also very easy to make mistakes. Read about them at http://www.lua.org/manual/5.1/manual.html#5.4.1
write your own small testprogram to try it out at http://www.lua.org/cgi-bin/demo

3.1.16 Table functions

The table functions do not support object oriented notation, but they are still useful.
`s=table.concat(t,[sep [, i [, j] ] ])` -- concatenate the members of t with the string sep in between. i and j can be used to offset start and limit length.
`table.insert(t [, pos], value)` -- inserts value at position pos in t, shifting up existing entries. If pos is omitted, it is appended at the end.
`value=table.remove(t[, pos])` -- remove t[pos] from table and shift down, returning the removed value.
`table.sort(t [, f])` -- sorts the elements in t. if the function f is provided it is used to compare items.

3.1.17 Mathematical functions

The mathematical functions do not support object oriented notation. There are many functions, but here are only these listed:
`a=math.abs(e)` -- the positive value of e
`a=math.ceil(e)` -- returns the smallest integer larger than or equal to e
`a=math.floor(e)` -- returns the largest integer smaller than or equal to e

3.1.18 Input and Output (IO)

The io libary supports object notification. It is used for reading/writing to files with buffered io. The most common ones:
`io.popen` -- this is blocked in AEP for CLxNX and not available on TH2.
`file=io.open(path [, op ])` -- get a file handle for path. Values for op:
`"r"` -- read (default)
`"w"` -- write
`"r+"` -- read/write, all previous data is preserved

"w+" -- read/write, all previous data is erased
"a","a+" -- append data, previous data is preserved, new data is appened at the endof the file

Sometimes you may see the letter 'b' appended. Read about it online.
file:read(...) -- read from the handle file. The patterns for ...
"*n" -- read a number (NB! "." is the decimal point)
"*l" -- the default pattern to read a line (ending with \n)
"*a" -- read the rest of the file (beware if the file is large)
0 -- used to detect EOF(end of file) (gives nil)
number -- read <number> bytes
file:write(...) -- write all arguments to file. The arguments must be strings or numbers.
file:close() -- close the file handle
file:flush() -- flush the pending output buffer (normally not needed)

3.1.19 Operating System functions

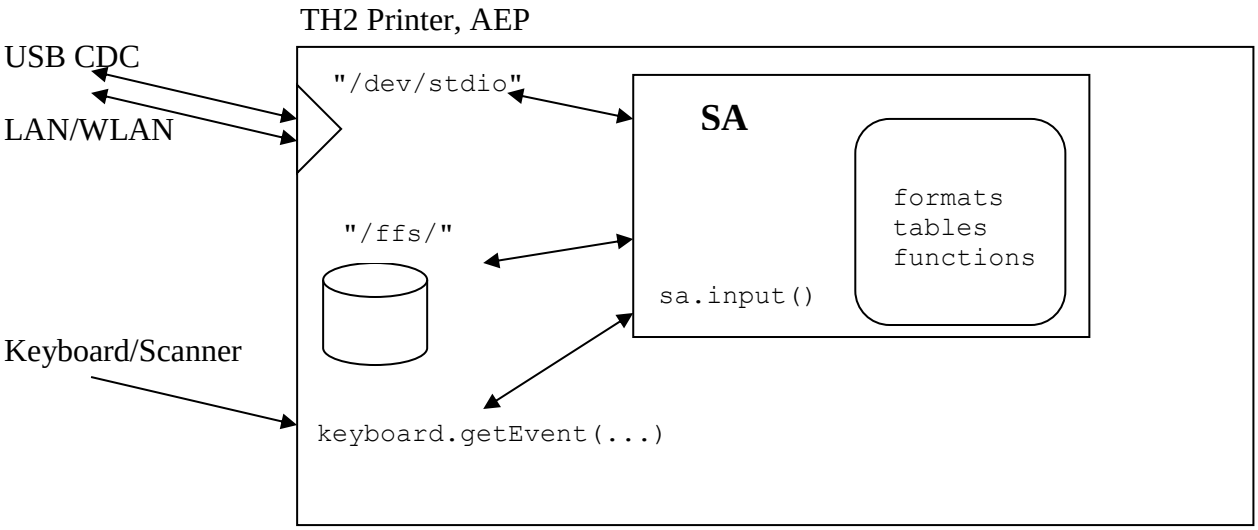
The OS-functions are very limited in TH2, and access is restricted in CLxNX
n=os.clock() -- returns uptime for aep
n=os.time([t]) -- returns the system time, or create a epoch time from the table t.
*=os.date([format [, time]]) -- date formatting, see system.ldateFormat for AEP
s=os.getenv("env") -- get the environment variable "env". Always nil in TH2.
err=os.execute("cmd") -- blocked in CLxNX and TH2 in AEP.
os.exit(x) -- this exits the process and is blocked in CLxNX and TH2.
os.remove(path) -- delete the file at path
os.rename(oldpath, newpath) -- rename a file
os.tmpname() -- creates a temp file and returns the (temporary) filename. Delete it!

4

System view

The Lua interpreter with the SATO extentions is called aep. The majority of aep applications running in SATO printers are the ones created in AEP Works. The AEP Works application runs in the SATO Standard Application, SA. SA runs in aep, commonly referred to as AEP.

4.1 The TH2 system view

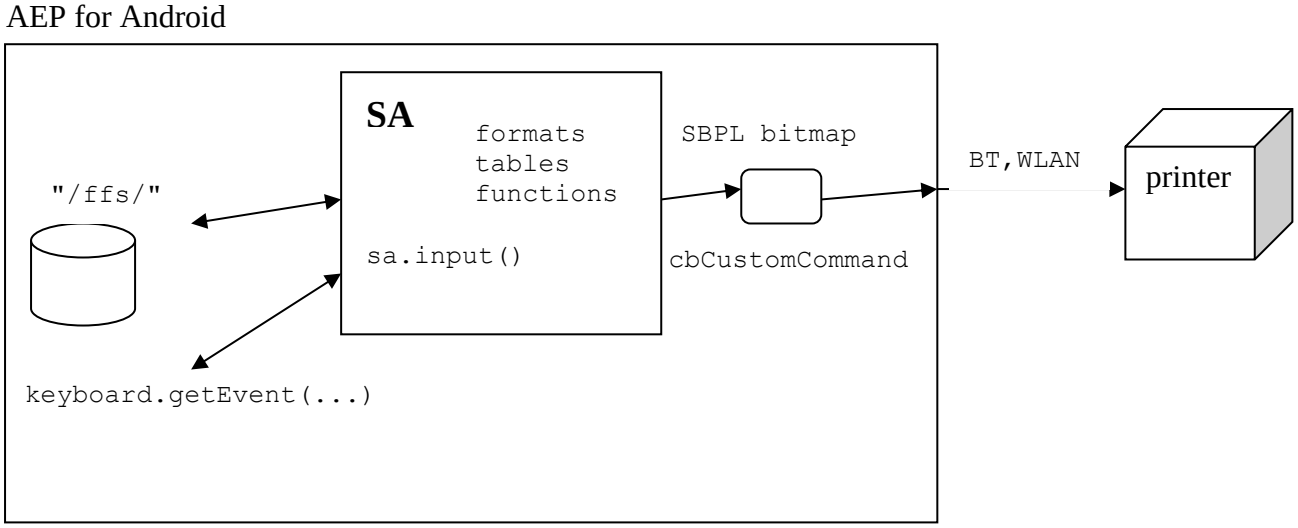


There's nofullfledged OS in the bottom. There are some devices, and an embedded filesystem for read and write at /ffs, and a temporary filesystem at /tmp. The SD-card slotgives access to a FAT16 2GB 8+3 UPPERCASE file system.

`"/dev/stdio"` is the path to read and write data sent from the host via USB CDC/COM: or TCP/IP port 9100/ftp/lpd.

`"/ffs/apps/sa/"`	Deleted & Installed at package install
`"/ffs/fonts"`	for Workspace fonts
`"/ffs/data"`	for data to retain between installs
`"/card/"`	SD-card, very slow
`"/tmp/"`	Temporary
`"/rom/"`	Read only filesystem

4.2 The AEP for Android system view



The AEP for Android system view is that it runs like PSim within an Android system, and the label image is rendered in the application. Before finalizing the print command, the Workspace application can append more data using the cbCustomCommand.

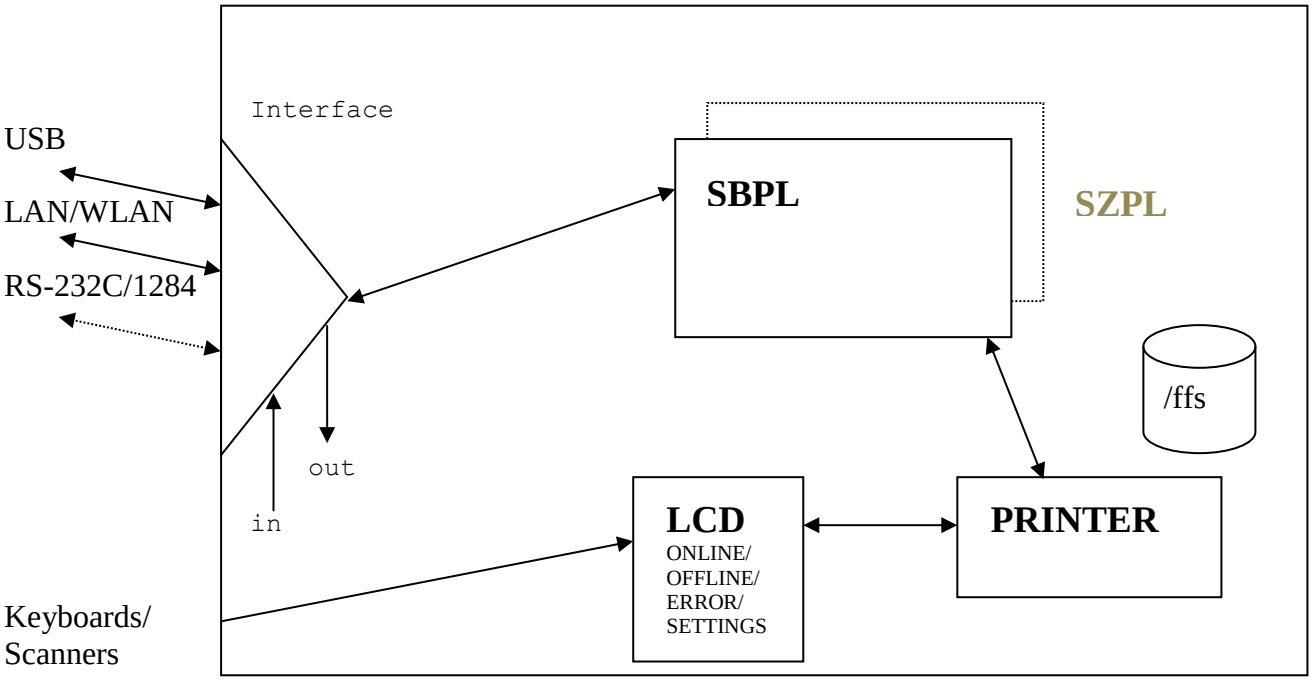
In AEP for Android the virtual /ffs is held in multiple instances, so /ffs is isolated per application.

"/ffs/apps/sa/"	Deleted & Installed at package install
"/ffs/fonts"	for Workspace fonts
"/ffs/data"	for data to retain between installs
"/card/"	SD-card, very slow
"/tmp/"	Temporary
"/rom/"	Read only filesystem

Virtual view of AEP for Android filesystem

4.3 The CLNX system view

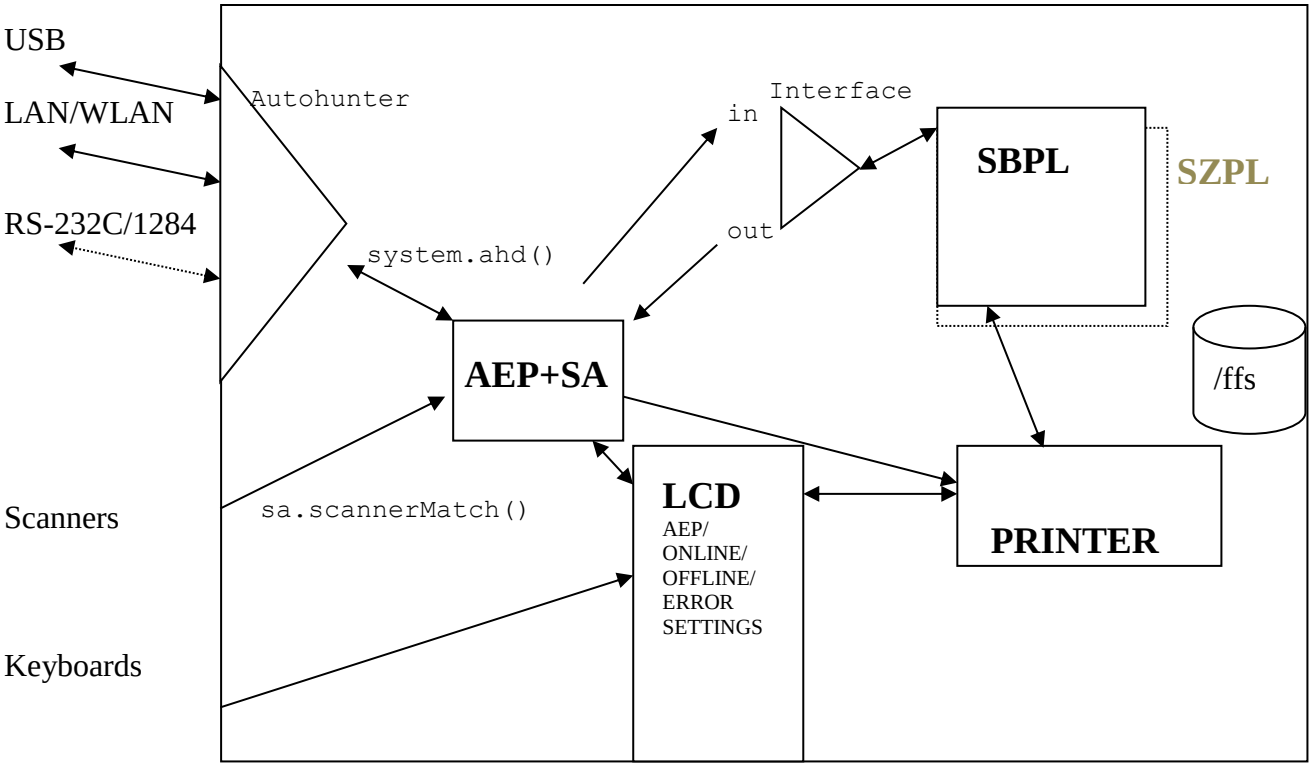
4.3.1 CLxNX system, Linux OS, AEP disabled



When AEP is disabled, the data is routed from the data interfaces into Interface process, and then forward onto the current protocol parser (e.g. SBPL). It will further process the data and finally request the rendered image buffer through the Printer process.
When keyboards or scanners are connected via USB, they are automatically routed to the LCD-process.

The LCD-process shows the various operating states.

4.3.2 CL4NX system, Linux OS, AEP enabled



When AEP is enabled, the data is routed from the data interfaces into the Autohunter process and Interface process can only access the in/out pipes, which will be forwarded to the current emulator (e.g. SBPL). The AEP application SA can intercept the data from system.ahd() and modify it before throwing it away or forwarding it to Interface via pipe in. AEP can read the output from the pipe out endpoint.

The LCD-process displays the various SA input screens in AEP-mode, and the other operating states as before.

When typical AEP Works applications are run in AEP+SA, the printing requests are sent directly to the PRINTER process. When the feature job.runSbplFromAep() is used the aep and SBPL process work together to build the print image, and SBPL will request the print process to print it.

When keyboards or are connected via USB, they are automatically routed to the LCD-process, unless sa.scannerMatch() claims ownership. The default behavior for sa.scannerMatch() is to only claim ownership for USB-devices that contain "scanner" in the device description.

Interface vs Autohunter
There are incompabilities between Interface and Autohunter.
Interface support unidirectional 1024(inbound)/1025(outbound).

Autohunter supports bidirectional 1024(inbound/outbound).
Autohunter does not support 1025 at all.
Interface guarantees to send query-response from port X to port X.
Autohunter does not guarantee that. It shouldn't be a problem in most cases.
Autohunter reads from one port at a time, and will stick to that port as long as data is within 5s from the previous chunk. This is a problem when used with AiOT, that tend to spam the network with ENQ-requests.
Interface inhibits TCP-queuing; this can cause data loss
Autohunter inhibits TCP-queuing by default (port type INET) which can cause data loss, but it can be reconfigured with system.ahd() (port type INET2).
Interface reads data into local buffers that may cause buffer overflow and disables flow-control.
Autohunter leaves data in the kernel driver buffers and withstands the SATO way of buffer overflow. Autohunter might be a bit more sensitive to RS-232C overflow if handshaking is disabled.

`"/ffs/apps/sa/" aka /mnt/data/ffs...`	Deleted & Installed at package install
`"/ffs/fonts"`	for Workspace fonts
`"/ffs/data"`	for data to retain between installs
`"/media/usb_front/"`	USB ms front
`"/media/usb_rear/"`	USB ms rear
`"/tmp/" aka /var/volatile/tmp...`	Temporary
`"/mnt/data/user/[sbpl|szpl|...]"`	Storage area base for emulators
`"/rom/"`	Another path in the filesystem

In addition to this the embedded Linux OS consists of a root filesystem which the technically endevoured can read about online.

4.4 General

After inserting a USB drive it will be automatically mounted. The below table lists the path(s) that are available in each model. The device can also be referenced using its name, e.g "/media/usb_rear2" -> "/media/MY_DRIVE".

CLNX Series	
/media/usb_rear	
/media/usb_front	
PW2NX	
/media/usb_rear	
FX3-LX	
/media/usb_rear1	
/media/usb_rear2	
/media/usb_internal	
CT4-LX	
/media/usb_rear2	
/media/usb_internal	
/media/usb_on_module	Available on WLAN module (option).

5

Programming Conventions

5.1 Non exposed functions

Function and variable names (tables, strings, etc.) not to be exposed to users outside SATO (hidden) should start with a '_'. All Lua functions and variables mentioned in this document should be preloaded before any user Lua application, unless other is stated.

5.2 Application naming restrictions

It is not recommended to alter any by SATO or by Lua (standard distribution) created tables. To prevent application naming collisions on the global scope with firmware, the following naming restrictions apply.

Table 1. Global scope name restrictions

Global name	Description
_	All names starting with a _ (underline). The _ by itself is legal.
barcodeObject, boxObject, circleObject, config, configTbl, device, display, ellipseObject, engine, errno, fs, gridObject, i18nObject, i18nStringObject, imageObject, job, keyboard, labelObject, lineObject, sdb, support, textBMObject, textBoxObject, textTTObject, shell, system, wlan, sato and aep.	By SATO reserved names.
socket, mime, ltn12, profiler, lxp and bit	By SATO included additional Lua libraries (configuration dependent).

5.3 Naming convention

Functions, variables etc. - camelCase
Constants - UPPERCASE

5.4 Function return parameters

All functions shall return an additional error number (see 7.2). Normal parameters shall indicate nil if failure so caller knows when to act on the returned error number. Example:

```
result, err = foo()
if result == nil then
  • We have an error, err contains the cause.
else
```

- `Everything is ok`

6

Application start

How a user Lua application is downloaded and started is described in ref [1].

6.1 CLxNX AEP applications

The CLxNX is designed with Linux, and the AEP application share the resources together with the other applications and system services making up the printer.
To get good performance the preferred design is to write event-driven applications.

7

Functions

The function section contains many implementation examples; those are to be treated as examples not absolute for implementation.

7.1 Files API

See ref [1], for printer file system structure.

Files from Lua should be accessed by use of the built-in "io", "os", and "system" libraries. Special file system functions like directory listing, formatting, etc. should be implemented in a new library called fs (file system).

The return values of the following functions are `nil, <err>` on failure and `<retval>, errno.ESUCCESS` on success. `<retval>` is the described return value. Unless otherwise noted, `<retval>` is `true`. `<err>` is an error number as defined in chapter 7.2, "Errors".

Paths to files and directories can either be absolute or relative. Relative paths will be appended to the current directory to form the full path. Directory paths can, optionally, end in a '/'. The directory separator is '/'. ('\' is used as an escape character in Lua strings; additionally, '/' is used in URLs, accepted in Windows as well as Linux/Unix). '.' refers to the directory itself, and ".." refers to the parent directory.

File and directory names are limited to 100 bytes in length; individual file systems may truncate a longer name (/ffs, /card), or disallow it to be created. A total path cannot be longer than 255 bytes. File and directory names may not contain ASCII '/' (value 0x2f) characters or NUL bytes (0x00).

For the SD Card, FAT16 limits apply (short file names, only uppercase characters, more disallowed characters).

Access rights - normal read/write/execute rights apply for the file system. To create a file or directory, the user must have write access to the parent directory.

`fs.format(<path>)`
Formats the file system <path>. This command is only valid for FAT file systems (i.e. the SD Card). Reinitializes the card and erases all files and directories. Access rights do not apply. If applied on another file system than the SD Card, this is equivalent to `fs.remove(<path>, true)`.

`fs.dir(<path>)`
Returns a Lua iterator over the entries of a given directory. Each time the iterator is called it returns a string containing the next entry of <path>; `nil` is returned when there are no more entries. "." and ".." are included and entries that are directories will have a "/" appended to their names.

`fs.cd([<path>])`
When an argument is given, this function changes the current working directory to the given `path`. If no argument is given it returns a string with the current working directory.

`fs.mkdir(<path> [,<option>])`
Creates a new directory. The argument is the name of the new directory. If second argument is “p” parent directories will be created if not present.

`fs.copy(<source>, <destination>)`
Creates a copy of the file designated by `<source>` as `<destination>`. If `<destination>` is a file and already exists, it is overwritten. Destination can be a directory, in which case `<destination>/<source>` is created. If `<source>` is a directory its entire content (and sub-directories) is copied to `<destination>`, creating `<destination>` if it does not exist (otherwise creating `<destination>/<e>`, where `<e>` is the last path component of `<source>`).

`fs.remove(<path> [, <recursive>])`
Removes an existing directory or file. If a directory is not empty, `fs.remove` will fail, unless the second argument is `true`, in which case the directory and all entries in it will be removed. Without the second argument it is equivalent to `os.remove()`, except for the return value. Default value for the second argument is `false`.

`fs.devInfo(<path>)`
Returns a table with information about the device (or "file system") that `<path>` resides on:
`used` – number of bytes used on device.
`free` – number of bytes free to use on the device.
`device` – name of the topmost node of the device.

Example:
```
a = fs.devInfo("/ffs")
```

```
for i,j in pairs(a) do print(i,j) end
used    108544
device  /ffs
free    8673280
```

`fs.stat(<path>)`
`fs.lstat(<path>)`
Returns a table with the file attributes corresponding to `<path>`. The attributes are as follows:
`type` - string representing the type of <path> (one of `file`, `directory`, `device`, `symbolic link`, or `unknown`).
`uid` - name of owner (`root`, `admin`, `manager`, or `user`)
`size` - file size, in bytes.
`modified` - time of last modification in `os.time()` units.
`name` – fully qualified path of file/directory.
`access` - access rights. A table with the following attributes:
`user` string containing any or none of the characters 'r', 'w' and 'x' once. 'r' signifies read access, 'w' write, and 'x' execute access for the owner.
`others` same as for "user", but describes the access rights for users other than the owner.

Important notes for fs.lstat():
- fs.lstat() is not available in TH2. It's the same as fs.stat() except that if `<path>` is a symbolic link, it will stat the link and not what it points to.
- If you have a link to a directory (link -> dir, note that fs.lstat("link") will give info about the link, but that fs.lstat("link/") will give info about dir.

`fs.access(<path>, <uid>[, <uaccess>[, <oaccess>]])`
Changes the access rights for the file or directory `<path>`. `<uid>` is the user name, `<uaccess>` is a string containing the owner's rights, and `<oaccess>` other user's rights. If argument 2, 3 or 4 is `nil`, the corresponding attribute will not be altered. Only the characters r, w, and x have effect, any case is accepted.
Not supported by all file systems. Changing `uid` on a file/directory on `/tmp` or `/card` has no effect.

`<retval>, <errno> = fs.chksum(<path> [,"ADLER" | "RIPEMD"])`
Returns a checksum (string with a hexadecimal number) on the file contents. The second argument selects the kind of checksum, with `"ADLER"` being the default.
`"ADLER"` gives an Adler-32 checksum and `"RIPEMD"` gives a RIPEMD-160 hash.
Adler-32 is a 32-bit checksum that may be used to detect accidental changes to a file. RIPEMD-160 is a 160-bit cryptographic hash. Adler-32 is significantly faster than RIPEMD-160, but has a greater risk of collisions and is susceptible to intentional alterations.

`fs.sync()`
Calls Linux sync command to write filesystem caches to disk. Recommended for removable media.

`rp[,err]=fs.realpath(path)`
Returns the canonicalized absolute pathname for the existing path.

7.2 Errors

Error numbers are stored in the Lua table `errno`. Symbolic names (`errno.ECOVEROPEN`) should be used and not their numerical value, in case the numbers change. The symbolic name, as well as an explanatory string is included in the errno table. `errno[errno.EACCES]` is the string "EACCES", and `errno.text(errno.EACCES)` is the string "Permission denied".

The available errors are listed below:

x	errno[errno.x]	errno.text(errno.x)
EACCES	"EACCES"	Permission denied
EBADF	"EBADF"	Bad file descriptor
EBATTLOW	"EBATTLOW"	Battery low
EBUSY	"EBUSY"	Device or resource busy
ECOPY_SRCISDST	"ECOPY_SRCISDST"	Destination is source
ECOVEROPEN	"ECOVEROPEN"	Cover open
ECUTTERSTART	"ECUTTERSTART"	Cutter error
ECUTTERSTUCK	"ECUTTERSTUCK"	Cutter error
EDOM	"EDOM"	Argument out of function's domain
EEXIST	"EEXIST"	File exists
EFAULT	"EFAULT"	Bad address
EGAPTOOLONG	"EGAPTOOLONG"	Gap too long
EIMARKTOOLONG	"EIMARKTOOLONG"	I-mark too long
EINVAL	"EINVAL"	Invalid argument
EIO	"EIO"	I/O Error
EISDIR	"EISDIR"	Is a directory
EMFILE	"EMFILE"	Too many open files
ENAMETOOLONG	"ENAMETOOLONG"	Filename too long
ENODEV	"ENODEV"	No such device
ENOENT	"ENOENT"	No such file or directory
ENOGAP	"ENOGAP"	Gap not found
ENOIMARK	"ENOIMARK"	I-mark not found
ENOMEM	"ENOMEM"	Not enough space
ENOPAPER	"ENOPAPER"	Out of paper
ENOSPC	"ENOSPC"	No space left on device
ENOTCONN	“ENOTCONN”	Not connected
ENOTDIR	"ENOTDIR"	Not a directory
ENOTEMPTY	"ENOTEMPTY"	Directory not empty
ENOTFOUND	"ENOTFOUND"	Not found
ENOTSUP	"ENOTSUP"	Not supported
EPARAM	"EPARAM"	Invalid argument
EPERM	"EPERM"	Operation not permitted
EPITCHERROR	"EPITCHERROR"	Pitch error
ERANGE	"ERANGE"	Result too large

EREINDEX	"EREINDEX"	Database error
EROFS	"EROFS"	Read-only file system
ESUCCESS	"ESUCCESS"	No Error
EXMLSCHEMA	"EXMLSCHEMA"	XML XSD error

7.3 Rendering interface

This chapter defines a rendering interface for Lua.

The goal is that GMCs and customers shall be able to use this interface when writing Lua application code.

The main idea of the rendering interface is to create text fields, barcode fields and graphic fields. These fields are objects with a number of properties that can be manipulated by methods. The text, barcode and graphic objects are added to a label object which can be printed.

7.3.1 General considerations

7.3.1.1 Type

All objects support the Lua built in functions type([var]) and tostring([var]).
type([var]) always returns "userdata" and tostring([var]) returns the object type, ex: "textTTObject".

7.3.1.2 Clip

Objects that are positioned completely or partly outside the printable area of a label will not cause an error. This means that it is possible to e.g. write a text field where only half of it is actually printed on the label.

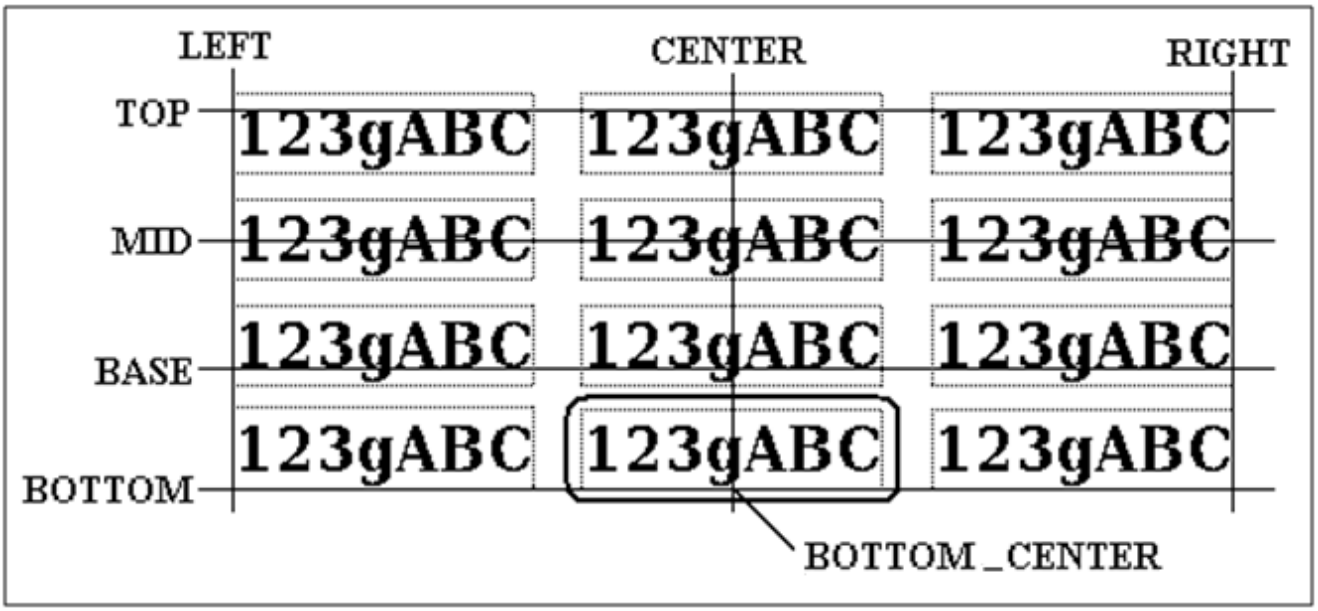
There are, however, special rules for barcodes.
A one dimensional barcode (for example EAN8) is only printed if the entire barcode fits on the printable area.
A two dimensional barcode (for example QR Code) is printed if the upper left corner of the barcode is positioned on the printable area.
The human readable text, if applicable, may or may not be printed if a barcode is positioned outside the printable area.

7.3.1.3 Positioning

The position of text, text box, barcode, box, circle, ellipse, grid, and image objects is set by a horizontal and a vertical position parameter (hPos and vPos). This position defines the position of the object anchor point. The anchor point can be set to one of twelve alternatives:

“TOP_LEFT”, “TOP_CENTER”, “TOP_RIGHT”,
“MID_LEFT”, “MID_CENTER”, “MID_RIGHT”,
“BASE_LEFT”, “BASE_CENTER”, “BASE_RIGHT”,
“BOTTOM_LEFT”, “BOTTOM_CENTER”, and “BOTTOM_RIGHT”.

The figure below shows the anchor points for a text field. The marked field has anchor point “BOTTOM_ CENTER”.



“LEFT”, “CENTER”, “RIGHT”, “TOP”, "MID", and "BOTTOM", are self-explanatory. The definition of “BASE”, however, is not quite as obvious.
This is how "BASE" is defined for different kinds of objects:

Text
The bottom row of capital letters. (See figure above.)

Barcodes
If no Human Readable field (HR) is printed, then "BASE" is the same as "BOTTOM".
If HR is printed below the bars, then "BASE" is at the bottom of the bars (above the HR).
If HR is printed above the bars, then "BASE" is at the bottom of the HR (above the bars).

Text box, Box, Circle, Ellipse, Grid, and Image
“BASE” is the same as "BOTTOM".

7.3.1.4 Codepage

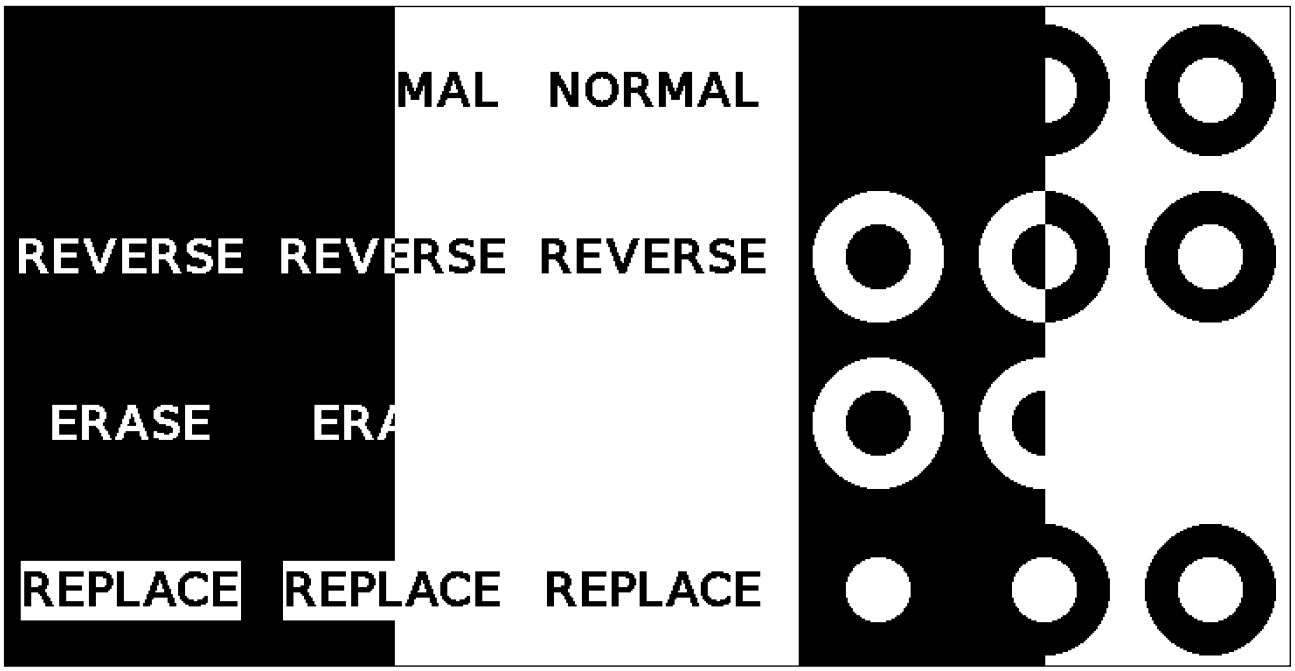
Texts using the Unicode codepage selection should be encoded as UTF-8.

7.3.1.5 Pen

All render objects have a parameter named pen. This parameter sets the rendering mode of an object. Pen can have four values:
NORMAL
Print with a black pen. This is the default mode.
REVERSE
Change the colour of the background when printing.
ERASE
Print with a white pen.

REPLACE
Erase background before printing the object.
Note: Printing a barcode with pen set to REPLACE does not necessarily mean that it can be scanned. There may still be obstructing objects printed to close to the barcode.

The figure below shows text fields and circles that are printed on black and white background with pen set to NORMAL, REVERSE, ERASE, and REPLACE respectively.



7.3.2 Text true type fields

TrueType (but not Truetype Collections) and also OpenType fonts that have encoding prioritized in order Unicode, Wansung, GB2312, BIG5, SJIS or Apple Roman can be used with textTTObject.

7.3.2.1 Constructor

new()

```
text, error = textTTObject.new([font[,text[,hPos[,vPos[,size[,codepage[,anchor
[,dir[,shear[,pen[,face[,embolden[,oblique[,shaper]]]]]]]]]]]]])
```

new() creates a textTTObject.
If all parameters are left out the default for each parameter will be used. Specific parameter(s) can be left out by writing nil at the parameter position. The shaper parameter has no function if shaper isn't supported by firmware. Support is checked by testing if the shaper method is nil or not.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
font = "/rom/truetype/SatoSans.ttf"
```

```
text = ""
hPos = 1
vPos = 1
size = 12
codepage = "UTF-8"
anchor = "TOP_LEFT"
dir = 0
shear = 0
pen = "NORMAL"
face = 1
embolden = false
oblique = false
shaper = true
```

If size is given as a table, it is interpreted as pixels and the default value is 12 for {} and {x,x} for {x}.

7.3.2.2 Methods

font()

```
font|error = <textTTObject>:font([font])
```

font is file path to the TT-font to be selected and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current font path is returned.
If a relative path is used (no leading /), the system will search in the following directories for the font (in order): `./` (current directory), `/card/FONTS`, `/ffs/fonts/`, and finally, `/rom/truetype`.
See Font resources for extended API functionality.

text()

```
text|error = <textTTObject>:text([text])
```

text is string to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current string is returned.

pos()

```
hPos,vPos|error = <textTTObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

size()

```
size|error = <textTTObject>:size([size])
```

size is font size in points to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current size is returned. Method size() can also be given a table to set the size in pixels with different size scaling for width and height.
Example:

```
--[[ normal 16 pointsize ]]--
ttObject = textTTObject.new(nil,nil,100,100,16)
assert(type(ttObject:size()) == "number"))
--[[ narrow width ]]--
ttObject = textTTObject.new(nil,nil,100,100,{24,32})
```

```
ttObject = textTTObject.new(nil,nil,100,100,{32,32})
ttObject:size({32,16})
assert(type(ttObject:size()) == "table"))
```

codepage()

```
codepage|error = <textTTObject>:codepage([codepage])
```

codepage is codepage to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current codepage is returned.

```
Codepages:
"DOS-858"
"ISO-8859-1"
"ISO-8859-2"
"ISO-8859-9"
"DOS-737"
"DOS-855"
"DOS-864"
"DOS-850"
"DOS-852"
"DOS-857"
"DOS-866"
("Windows-932")
"Windows-1250"
"Windows-1251"
"Windows-1252"
"Windows-1253"
"Windows-1254"
"Windows-1255"
"Windows-1256"
"Windows-1257"
"IBM CP 00869"
"DOS-862"
"UTF-8"
```

anchor()

```
anchor|error = <textTTObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <textTTObject>:dir([dir])
```

dir is text printing direction [0..359] to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

shear()

```
shear|error = <textTTObject>:shear([shear])
```

shear is used to get an Italic style on the text were parameter is in percentage. 100 is one font width of shear [-150..150] to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

pen()
`pen|error = <textTTObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <textTTObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

lineSpacing()
`space, error = <textTTObject>:lineSpacing()`
Return line spacing for selected font and size. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

fit()
size, error = `<textTTObject>:fit([width[,height]])`
Return the best matching point size for specified width and height. Width should be in the range [n - $2*10^9$] ('n' is twice the width of the widest character in selected font and size) and height in [8 – $2*10^9$]. nil can be used at any position and then the default value is used ($2*10^9$). Maximum reported point size is 128. error equals errno.ESUCCESS on OK, else errno.EPARAM.

face()
`index, error = <textTTObject>:face([index])`
Set font face index to be used. Error is set to errno.EPARAM if invalid face index and retuned index nil. On success returns new index and error nil. The number of faces in a font is the number of indexes in the returned font information table by method info().
If called without parameter current face index is returned.
See Font resources for extended API functionality.

embolden()
`embolden, error = <textTTObject>:embolden([embolden])`
Enable|disable embolding (make font Bolden). Error is set to errno.EPARAM if not boolean and embolden nil. On success returns new embolden and error nil.

oblique()
`oblique, error = <textTTObject>:oblique([oblique])`
Enable|disable oblique (make font Italic). Error is set to errno.EPARAM if not boolean and oblique nil. On success returns new oblique and error nil.

shaper()

```
shaper, error = <textTTObject>:shaper([shaper])
```

This function is nil when shaper isn't supported. Enable|disable shaper (text shaping using HarfBuzz). Error is set to errno.EPARAM if not boolean and shaper nil. On success returns new shaper and error nil. Returns additional cluster table if called with nil and shaper enabled.
Eg.

```
-- "น้ำปู"
str = ""
_str = [[E0 B8 99 E0 B8 B1 E0 B9 89 E0 B8 9B E0 B8 B9 E0 B9 88]]
for x in _str:gmatch("(%S+)") do str=str..string.char(tonumber(x,16)) end
tt = textTTObject.new(nil,str)
_, cluster = tt:shaper()
print((json.encode(cluster)))
[1,1,1,4,4,4]
```

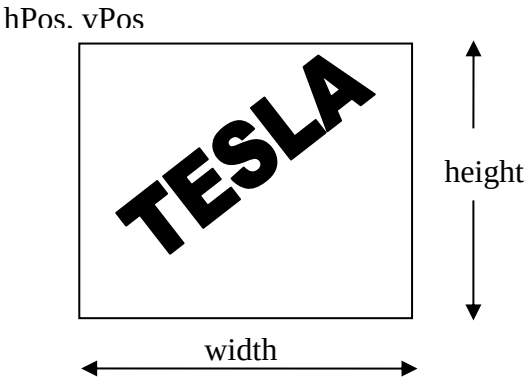
info()

```
info, error  = <textTTObject>:info([index])
```

Return true type font information. error equals errno.ESUCCESS on OK, else errno.EPARAM. If called without parameter the returned table has one index per font/face in file. If called with parameter only the indexed font/face table is returned. Each index has the following fields where numerical fields are relative base line and scaled according to selected size.

Eg:

```
["Postscript"]={
["underlineThickness"]=1,
["underlinePosition"]=-2,
}
["OS2"]={
["usWinDescent"]=7,
["sTypoAscender"]=25,
["usWinAscent"]=30,
["yStrikeoutPosition"]=8,
["sTypoDescender"]=-7,
}
["styleName"]="Regular"
["familyName"]="Sato Sans"
```



clone()

```
clone, error = <textTTObject>:clone()
```

clone() creates an exact copy of the original textTTObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()

```
tbl = textTTObject.ranges()
```

Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. The table are arranged so that every attribute is a key and the value for that key is another table containing the ranges and options.

The different keys that can be found in the attributes sub-table:

```
min         -The minimum value or the minimum length of a string
max         -The maximum value or the maximum length of a string
type        -The type of the attribute, number, string or bool
options     -An array of the options that are supported for this attribute
```

Example:

```
> obj = textTTObject.new ()
> return (json.encode(obj.ranges(),nil,true))
{
 "vPos":{
  "type":"number"
 },
 "hPos":{
  "type":"number"
 },
 "dir":{
  "type":"number"
 },
 "text":{
  "type":"string"
 },
 "codepage":{
  "options":["DOS-858","ISO-8859-1","ISO-8859-2","ISO-8859-9","DOS-737",
   "DOS-855","DOS-864","DOS-850","DOS-852","DOS-857","DOS-866",
   "Windows-932","Windows-1250","Windows-1251","Windows-1252",
```

```
   "Windows-1253","Windows-1254","Windows-1255","Windows-1256",
   "Windows-1257","IBM CP 00869","DOS-862","UTF-8"],
  "type":"string"
 },
 "pen":{
  "options":["NORMAL","REVERSE","REPLACE","ERASE"],
  "type":"string"
 },
 "anchor":{
  "options":["TOP_LEFT","TOP_CENTER","TOP_RIGHT","MID_LEFT","MID_CENTER",
   "MID_RIGHT","BASE_LEFT","BASE_CENTER","BASE_RIGHT","BOTTOM_LEFT",
   "BOTTOM_CENTER","BOTTOM_RIGHT"],
  "type":"string"
 },
 "shear":{
  "min":-150,
  "type":"number",
  "max":150
 },
 "face":{
  "type":"number"
 },
 "size":{
  "min":1,
  "type":"number",
  "max":128
 }
}
```

7.3.3 Text bitmap fields

7.3.3.1 Constructor

new()

```
text, error = textBMObject.new([font[,text[,hPos[,vPos[,hMag[,vMag[,codepage
                                [,anchor[,dir[,pen]]]]]]]]])
text, error = textBMObject.new([font[,text[,hPos[,vPos[,hMag[,vMag[,codepage

[,anchor[,dir[,pen[,fontset[,vertically[,typeface[,bold[,smooth[,equalSpacing]]]
]]]]]]]]]]]])
```

new() creates a textBMObject.
If all parameters are left out the default for each parameter will be used. Specific parameter(s) can be left out by writing nil at the parameter position.
The arguments from fontset, are only available after require "lsrender".
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Downloaded bitmap fonts can also be rendered given that they can be identified and read by the font as a file path. Downloaded bitmap fonts remain in memory until they are disposed, see dispose().

Default values:
```
font = "X1"
text = ""
hPos = 1
vPos = 1
hMag = 1
vMag = 1
codepage = "UTF-8"
anchor = "TOP_LEFT"
dir = 0
pen = "NORMAL"
```

Default values (with require("lsrender")):
```
fontset = "X208"
vertically = false
typeface = "Gothic"
bold = false
smooth = false
equalSpacing = false
```

dispose()
```
cnt[, error] = textBMObject.dispose(font)
```
dispose() frees up memory resources for unused downloaded bitmap fonts. font is either a file path that used when loading the font (see new()) or a boolean with value true for disposing all unused bitmap fonts. NB! The font is still in use if a textBMObject, textBoxObject or labelObject has a reference to it. The number of disposed fonts is returned in cnt or nil and error code on error.

7.3.3.2 Methods

font()
```
font|error = <textBMObject>:font([font])
```
font is font `["M"|"OCR-B"|"POP1"|"POP2"|"POP3"|"PRICE"|"S"|"U"|"XU"|"X1"|"X2"|"X3"]` to be selected and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current font is returned. Japanese firmware contains K16, K24 and K32 as resident Kanji bitmap fonts.

The fonts POP1, POP2, POP3 and PRICE contain only digits, some punctuations and monetary symbols. OCR-B contains digits and uppercase letters. The fonts M, U, S, X1, X2, X3 and XU contain non-graphical symbols defined in WGL4.

Downloaded bitmap fonts can also be set using the font method, in which font is the file path where to read it. If a relative path is used (no leading /), the system will search in the following directories for the font (in order): `./` (current directory), `/card/FONTS`, and `/ffs/fonts/`.

text()
```
text|error = <textBMObject>:text([text])
```
text is string to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current string is returned.

pos()

```
hPos,vPos|error = <textBMObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

mag()

```
hMag, vMag|error = <textBMObject>:mag([hMag[,vMag]])
```

hMag,vMag are horizontal/vertical pixel magnification to set [1..12], error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hMag and vMag are returned.

codepage()

```
codepage|error = <textBMObject>:codepage([codepage])
```

codepage is codepage to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current codepage is returned.

```
Codepages:
"DOS-858"
"ISO-8859-1"
"ISO-8859-2"
"ISO-8859-9"
"DOS-737"
"DOS-855"
"DOS-864"
"DOS-850"
"DOS-852"
"DOS-857"
"DOS-866"
("Windows-932")
"Windows-1250"
"Windows-1251"
"Windows-1252"
"Windows-1253"
"Windows-1254"
"Windows-1255"
"Windows-1256"
"Windows-1257"
"IBM CP 00869"
"DOS-862"
"UTF-8"
```

anchor()

```
anchor|error = <textBMObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()
`dir|error = <textBMObject>:dir([dir])`
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

pen()
`pen|error = <textBMObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <textBMObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

lineSpacing()
space, error = `<textBMObject>:lineSpacing()`
Return line spacing for selected font and magnification. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <textBMObject>:clone()`
clone() creates an exact copy of the original textBMObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <textBMObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

fontset()
`fontset | error = <textBMObject>:fontset([fontset])`
Sets the fontset. Depends on require("lsrender").

vertically()
`vertically | error = <textBMObject>:vertically([vertically])`
Sets the vertically property. When true, it renders vertically. Depends on require("lsrender").

typeface()
`typeface | error = <textBMObject>:typeface([typeface])`
Sets the typeface. Depends on require("lsrender").

bold()
```
bold | error = <textBMObject>:bold([bold])
```
Sets the bold property. When true, it renders the font fatter. Depends on require("lsrender").

smooth()
```
smooth | error = <textBMObject>:smooth([smooth])
```
Sets the smooth property. When true, it extrapolates pixels to smoothen the ridges. Depends on require("lsrender").

equalSpacing()
```
equalSpacing | error = <textBMObject>:equalSpacing([equalSpacing])
```
Sets the equalSpacing property. When true, it renders the font in monospace fashion. Depends on require("lsrender").

7.3.4 Text box fields

7.3.4.1 Constructor

new()
box,error = textBoxObject.new(obj[,width[,rows [,delimiter[,hyphen[,align
[,margin[,style[,fit[,height[,wrapChars]]]]]]]]])
new() creates a textBoxObject. If all parameters are left out the default for each parameter will be used. Specific parameter(s) can be left out by writing nil at the parameter position. error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. Detailed parameter description can be found in the methods section 7.3.4.2. Soft wrap points are <space> or <tab> characters. Soft wrap points are always printed unless at the beginning or at the end of a row, then they are removed.

Default values:
```
width = 448           -- Full print width (printer dependant)
rows = 0              -- infinite
delimiter = "\n"
hyphen = "-"
align = "TOP_LEFT"
margin =
{top=0,bottom=0,left=0,right=0}
style = "NORMAL"
fit = false           -- do not autosize
height = nil          -- Size according to rows and size of text
wrapChars = "\032\009\227\128\128\227\128\129" -- Space, tab, wide space, and
                                                  wide comma
```

7.3.4.2 Methods

object()
object|error = <textBoxObject>:object([object])
object is the object to be used by the box, error is to errno.ESUCCESS if OK, otherwise errno.EPARAM. object can be of type textTTObject or textBMObject. When set an internal copy of

the object is created. All object drawing constraints are used by the box object (hPos,vPos, dir and anchor). If called without parameter a reference to the current object is returned.
Limitation: object must be textTTObject type if fit is true, else error will be returned.

width()
width|error = <textBoxObject>:width([width])
width is the box width to set, error is to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current width is returned. The width parameter must be >=n where 'n' is twice the width of the widest character in the selected font and size.

rows()
rows|error = <textBoxObject>:rows([rows])
For fit=false, rows is the (maximum) number of rows in the box, error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM.
rows set to 0 makes the box have as many rows as is needed to fit the text. If called without parameters current rows is returned.
The value of rows is ignored if height is set (not 0 or nil), except when rows is 1 and fit is true; then there will only be 1 row.
For fit = true, rows gives the maximum height (excluding top and bottom margins) of the box by: rows * lineSpacing() of the truetype object (infinite if rows = 0).

delimiter()
delimiter|error = <textBoxObject>:delimiter(delimiter)
delimiter is a Lua pattern matching hard wrap points, error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. Wrapping will always occur at this position. Any found <CR> or <CR><LF> in the source string will be replaced by a <LF> before delimiter calculation. Delimiter characters are never printed. If called without parameter current pattern will be returned.

enableCache()
state,error = textBoxObject.enableCache([enable])
Controls if the textBox-cache is enabled or not. It is enabled by default. The previous state is returned in status; if the enable-parameter is omitted, the current state is returned. If the enable-parameter is invalid nil, errno.EINVAL is returned.

gc()
textBoxObject.gc()
Performs garbage collect on the textBox-cache. Is normally handled by the system.

hyphen()
hyphen|error = <textBoxObject>:hyphen([hyphen])
hyphen is the character that indicates that a word continues on the following line, error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. It's printed at the end of the line were the wrap occurred. Hyphen is only inserted if a word (word here is characters between two wrap points) is

longer than the box width. The hyphen can only be empty or one character long. If called without parameter current hyphen will be returned.

align()
align|error = <textBoxObject>:align([align])
align is one of `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"| "MID_CENTER"| "MID_RIGHT"|"BOTTOM_LEFT"| "BOTTOM_CENTER"|"BOTTOM_RIGHT"]`.
("BASE_XXX" will be accepted and interpreted as "BOTTOM_XXX".)
align is the alignment of the object within the box, error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameters current align is returned.
Note that it is object's anchor that defines the anchor point of the text box.

margin()
margin|error = <textBoxObject>:margin([margin])
margin is the margin of the box, error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameters current margin is returned. margin is a table of the format {top,bottom,left,right}, where keys are integer variables in the range [0..448].

style()
style|error = <textBoxObject>:style([style])
style is the style of the box, error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameters current style is returned. style can be ["NORMAL"|"INVERSE"].

fit()
fit|error = <textBoxObject>:fit([fit])
fit tells if the object should be scaled to fit within the box boundaries, error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameters current value is returned. fit can be [true|false].
The value of rows is ignored if height is set (not 0 or nil), except when rows is 1 and fit is true; then there will only be 1 row.
Fit will never increase the size of the text.
Limitation: object must be textTTObject type if fit is true, else error will be returned.

height()
height|error = <textBoxObject>:height([height])
height is the height of the text area in dots, error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameters current height setting is returned (not necessarily the same as the resulting height of the box). height set to 0 or nil disables height restriction by this parameter (instead it is defined by rows, text, text size, etc).
If height is set (not 0 or nil) the value of rows is ignored, except when rows is 1 and fit is true; then there will only be 1 row. Note that top and bottom margins add to the total height of the box.

wrapChars()
wrapChars|error = <textBoxObject>:wrapChars([wrapChars])

wrapChars is a string with characters. These characters defines suitable places to end a row in the textBox and start a new row if the string is wider than the textbox width. Error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter current wrapChars will be returned.

fieldSize()
`hPos,vPos,width,height,error = <textBoxObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

7.3.4.3 Summary of Fit, Rows, and Height

height = nil (or 0):
Box ends just after last line.
Maximum height of box is rows * lineSpacing() of truetype object (plus any top and bottom margins). Infinite if rows = 0.

height > 0:
height of the box is height (plus any top and bottom margins), even if text ends sooner.
rows has no effect, except when rows = 1 and fit = true.

fit = false
Text size will not change.

	height = nil/0	height > 0
rows = 0	Text may have any number of rows. All text is included.	Text may have any number of rows (that fit in the given height). May truncate text.
rows = 1	1 row. May truncate text.	
rows > 1	Number of lines <= rows. May truncate text.	

fit = true
Actual text size may be smaller than given in the textTTObject, but never larger.
All text is included if it is possible to fit, otherwise the box is not printed.
rows = 0, rows > 1: Text may have any number of rows.
rows = 1: Text will only have 1 row.

clone()
`clone, error = <textBoxObject>:clone()`
clone() creates an exact copy of the original textBoxObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <textBoxObject>:ranges()`

Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.5 Barcode EAN-8 fields

7.3.5.1 Constructor

newEan8()
bcEan8, error = barcodeObject.newEan8([data[,hPos[,vPos[,anchor[,dir[,height
[,narrowWidth[,humanReadable[,pen]]]]]]]])

newEan8() creates a barcodeEan8Object.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "12345670"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.5.2 Methods

data()

```
data|error = <barcodeEan8Object>:data([data])
```

data is the string containing the data for the barcode.
EAN-8 has seven digits and a check digit. If the string contains seven digits, a check digit is generated and added. If the string contains eight digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeEan8Object>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

`anchor|error = <barcodeEan8Object>:anchor([anchor])`
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()
`dir|error = <barcodeEan8Object>:dir([dir])`
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()
`height|error = <barcodeEan8Object>:height([height])`
height is barcode height in dots (1-999 excluding human readable text and "beard") to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()
`narrowWidth|error = <barcodeEan8Object>:narrowWidth([narrowWidth])`
narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

humanReadable()
`humanReadable|error = <barcodeEan8Object>:humanReadable([humanReadable])`
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
`pen|error = <barcodeEan8Object>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeEan8Object>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeEan8Object>:clone()`
clone() creates an exact copy of the original barcodeEan8Object.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeEan8Object>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.6 Barcode JAN-8 fields

JAN-8 (Japan Article Numbering) is an EAN-8 barcode with the two first digits set to 45 or 49, i.e. the country codes for Japan.
To avoid any compatibility problems, our implementation of JAN-8 will allow all valid EAN-8 data. The default data, however, is changed to a correct JAN-8 data.

7.3.6.1 Constructor

newJan8()
bcJan8, error = barcodeObject.newJan8([data[,hPos[,vPos[,anchor[,dir[,height
[,narrowWidth[,humanreadable[,pen]]]]]]]])

newJan8() creates a barcodeJan8Object.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "49012347"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.6.2 Methods

data()
```
data|error = <barcodeJan8Object>:data([data])
```
data is the string containing the data for the barcode.
JAN-8 has seven digits and a check digit. If the string contains seven digits, a check digit is generated and added. If the string contains eight digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()
```
hPos,vPos|error = <barcodeJan8Object>:pos([hPos[,vPos]])
```
hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()
```
anchor|error = <barcodeJan8Object>:anchor([anchor])
```
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()
```
dir|error = <barcodeJan8Object>:dir([dir])
```
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()
```
height|error = <barcodeJan8Object>:height([height])
```
height is barcode height in dots (1-999 excluding human readable text and "beard") to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()
```
narrowWidth|error = <barcodeJan8Object>:narrowWidth([narrowWidth])
```
narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

humanReadable()
```
humanReadable|error = <barcodeJan8Object>:humanReadable([humanReadable])
```
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
```
pen|error = <barcodeJan8Object>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeJan8Object>:fieldSize()
```

Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodeJan8Object>:clone()
```
Not available on TH2.
clone() creates an exact copy of the original barcodeJan8Object.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeJan8Object>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.7 Barcode EAN-13 fields

7.3.7.1 Constructor

newEan13()
bcEan13, error = barcodeObject.newEan13([data[,hPos[,vPos[,anchor[,dir[,height
[,narrowWidth[,humanreadable[,pen]]]]]]]])

newEan13() creates a barcodeEan13Object.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "1234567890128"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.7.2 Methods

data()
```
data|error = <barcodeEan13Object>:data([data])
```
data is the string containing the data for the barcode.
EAN-13 has twelve digits and a check digit. If the string contains twelve digits, a check digit is

generated and added. If the string contains thirteen digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()
`hPos,vPos|error = <barcodeEan13Object>:pos([hPos[,vPos]])`
hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()
`anchor|error = <barcodeEan13Object>:anchor([anchor])`
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()
`dir|error = <barcodeEan13Object>:dir([dir])`
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()
`height|error = <barcodeEan13Object>:height([height])`
height is barcode height in dots (1-999 excluding human readable text and "beard") to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()
`narrowWidth|error = <barcodeEan13Object>:narrowWidth([narrowWidth])`
narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

humanReadable()
`humanReadable|error = <barcodeEan13Object>:humanReadable([humanReadable])`
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
`pen|error = <barcodeEan13Object>:pen([pen])`

pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeEan13Object>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodeEan13Object>:clone()
```
clone() creates an exact copy of the original barcodeEan13Object.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeEan13Object>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.8 Barcode JAN-13 fields

JAN-13 (Japan Article Numbering) is an EAN-13 barcode with the two first digits set to 45 or 49, i.e. the country codes for Japan.
To avoid any compatibility problems, our implementation of JAN-13 will allow all valid EAN-13 data. The default data, however, is changed to a correct JAN-13 data.

7.3.8.1 Constructor

newJan13()
bcJan13, error = barcodeObject.newJan13([data[,hPos[,vPos[,anchor[,dir[,height
[,narrowWidth[,humanreadable[,pen]]]]]]]])

newJan13() creates a barcodeJan13Object.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "4901234567894"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
```

```
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.8.2 Methods

data()

```
data|error = <barcodeJan13Object>:data([data])
```

data is the string containing the data for the barcode.
JAN-13 has twelve digits and a check digit. If the string contains twelve digits, a check digit is generated and added. If the string contains thirteen digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeJan13Object>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeJan13Object>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeJan13Object>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()

```
height|error = <barcodeJan13Object>:height([height])
```

height is barcode height in dots (1-999 excluding human readable text and "beard") to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()

```
narrowWidth|error = <barcodeJan13Object>:narrowWidth([narrowWidth])
```

narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

humanReadable()

`humanReadable|error = <barcodeJan13Object>:humanReadable([humanReadable])`
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
`pen|error = <barcodeJan13Object>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeJan13Object>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeJan13Object>:clone()`
clone() creates an exact copy of the original barcodeJan13Object.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeJan13Object>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.9 Barcode UPC-A fields

7.3.9.1 Constructor

newUpca()
bcUpca, error = barcodeObject.newUpca([data[,hPos[,vPos[,anchor[,dir[,height
[,narrowWidth[,humanReadable[,pen]]]]]]]])

newUpca() creates a barcodeUpcaObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "123456789012"
hPos = 100
vPos = 100
```

```
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.9.2 Methods

data()

```
data|error = <barcodeUpcaObject>:data([data])
```

data is the string containing the data for the barcode.
UPC-A has eleven digits and a check digit. If the string contains eleven digits, a check digit is generated and added. If the string contains twelve digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeUpcaObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeUpcaObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeUpcaObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()

```
height|error = <barcodeUpcaObject>:height([height])
```

height is barcode height in dots (1-999 excluding human readable text and "beard") to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()

```
narrowWidth|error = <barcodeUpcaObject>:narrowWidth([narrowWidth])
```

narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

humanReadable()
`humanReadable|error = <barcodeUpcaObject>:humanReadable([humanReadable])`
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
`pen|error = <barcodeUpcaObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeUpcaObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeUpcaObject>:clone()`
clone() creates an exact copy of the original barcodeUpcaObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeUpcaObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.10 Barcode UPC-E fields

7.3.10.1 Constructor

newUpce()
bcUpce, error = barcodeObject.newUpce([data[,hPos[,vPos[,anchor[,dir[,height
[,narrowWidth[,humanreadable[,pen]]]]]]]])

newUpce() creates a barcodeUpceObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "1234565"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.10.2 Methods

data()

```
data|error = <barcodeUpceObject>:data([data])
```

data is the string containing the data for the barcode.
UPC-E has six digits and a check digit. If the string contains six digits, a check digit is generated and added. If the string contains seven digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeUpceObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeUpceObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeUpceObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()

```
height|error = <barcodeUpceObject>:height([height])
```

height is barcode height in dots (1-999 excluding human readable text and "beard") to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()

```
narrowWidth|error = <barcodeUpceObject>:narrowWidth([narrowWidth])
```

narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

humanReadable()
`humanReadable|error = <barcodeUpceObject>:humanReadable([humanReadable])`
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
`pen|error = <barcodeUpceObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeUpceObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeUpceObject>:clone()`
clone() creates an exact copy of the original barcodeUpceObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeUpceObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.11 Barcode Code 39, Code93 fields

7.3.11.1 Constructor

newCode39()
bcCode39, error = barcodeObject.newCode39([data[,hPos[,vPos[,anchor[,dir[,height
[,barRatio[,narrowWidth[,humanreadable[,pen]]]]]]]]])

newCode39() creates a barcodeCode39Object.

newCode93()
bcCode93, error = barcodeObject.newCode93([data[,hPos[,vPos[,anchor[,dir[,height

[,narrowWidth[,humanreadable[,pen]]]]]]]]])

newCode93() creates a barcodeCode93Object. It is available after require("lsrender").

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "*123456789*"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
barRatio = "1:2"
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

Default value Code93:
```
data = "123456789"
```

7.3.11.2 Methods

data()

```
data|error = <barcodeCode39Object|barcodeCode93Object>:data([data])
```

data is the string containing the data for the barcode.
Code 39 has a start character '*', one or more characters, and a stop character '*'. No check digit is included.
Code 39 can encode uppercase letters ('A' - 'Z'), numbers ('0' - '9'), and a handful of special characters ('-', '.', ' ', '$', '/', '+', and '%').
If a start character is not entered, start and stop character will be added.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.
In Code93, the start / stop character (*) are automatically added.

pos()

```
hPos,vPos|error = <barcodeCode39Object|barcodeCode93Object>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeCode39Object|barcodeCode93Object>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|`

"BOTTOM_CENTER"|"BOTTOM_RIGHT"] and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeCode39Object|barcodeCode93Object>:dir([dir])
```

dir is text printing direction [0|90|180|270] to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()

```
height|error = <barcodeCode39Object|barcodeCode93Object>:height([height])
```

height is barcode height in dots (1-999 excluding human readable text) to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

barRatio()

```
barRatio|error = <barcodeCode39Object>: barRatio ([barRatio])
```

barRatio is the bar ratio, i.e. the width ratio between a thin and a wide bar, to set ["1:3"|"2:5"|"1:2"]. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode ratio is returned. This is only supported in Code39.

narrowWidth()

```
narrowWidth|error =
  <barcodeCode39Object|barcodeCode93Object>:narrowWidth([narrowWidth])
```

narrowWidth is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.
Note that if barRatio is set to "2:5", it is not possible to use an odd value of narrowWidth. If an odd value is set, the value increased by one is used.

humanReadable()

```
humanReadable|error =
<barcodeCode39Object|barcodeCode93Object>:humanReadable([humanReadable])
```

humanReadable defines whether or not an interpretation line shall be printed [true|false]. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()

```
pen|error = <barcodeCode39Object|barcodeCode93Object>:pen([pen])
```

pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()

```
hPos, vPos, width, height, error =
<barcodeCode39Object|barcodeCode93Object>:fieldSize()
```

Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM. NB! barcodeCode93Object gives incorrect results.

clone()

```
clone, error = <barcodeCode39Object|barcodeCode93Object>:clone()
```

clone() creates an exact copy of the original barcodeCode39Object.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()

```
tbl = <barcodeCode39Object>:ranges()
```

Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.12 Barcode Codabar fields

7.3.12.1 Constructor

newCodabar()

```
bcCodabar, error = barcodeObject.newCodabar([data[,hPos[,vPos[,anchor[,dir
        [,height[,barRatio[,narrowWidth[,humanreadable[,pen]]]]]]]]])
```

newCodabar() creates a barcodeCodabarObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "A0123456789B"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
barRatio = "1:2"
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.12.2 Methods

data()

```
data|error = <barcodeCodabarObject>:data([data])
```

data is the string containing the data for the barcode.

Codabar has a start character, one to twenty characters, and a stop character. No check digit is included.
Codabar can encode numbers (0-9), and a handful of special characters ('. ', '+', ': ', '/', '$', and '-'). The start and stop character can be set to 'A', 'B', 'C', 'D', 'E', 'N', 'T', or '*'. Lower case versions of these characters are also accepted as start and stop characters.
If a start character is not entered, start ('A') and stop character ('B') will be added.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()
`hPos,vPos|error = <barcodeCodabarObject>:pos([hPos[,vPos]])`
hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()
`anchor|error = <barcodeCodabarObject>:anchor([anchor])`
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.
.

dir()
`dir|error = <barcodeCodabarObject>:dir([dir])`
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()
`height|error = <barcodeCodabarObject>:height([height])`
height is barcode height in dots (1-999 excluding human readable text) to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

barRatio()
`barRatio|error = <barcodeCodabarObject>: barRatio ([barRatio])`
barRatio is the bar ratio, i.e. the width ratio between a thin and a wide bar, to set ["1:3"|"2:5"|"1:2"]. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode ratio is returned.

narrowWidth()
`narrowWidth|error = <barcodeCodabarObject>:narrowWidth([narrowWidth])`
narrowWidth is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

Note that if barRatio is set to "2:5", it is not possible to use an odd value of narrowWidth. If an odd value is set, the value increased by one is used.

humanReadable()
```
humanReadable|error =
<barcodeCodabarObject>:humanReadable([humanReadable])
```
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
```
pen|error = <barcodeCodabarObject>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeCodabarObject>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodeCodabarObject>:clone()
```
clone() creates an exact copy of the original barcodeCodabarObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeCodabarObject>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.13 Barcode Bookland fields

Bookland is the five digit supplemental barcode that encodes the currency and price of a book. An EAN 13 barcode is also needed to encode the ISBN number.

7.3.13.1 Constructor

newBookland()
bcBookland, error = barcodeObject.newBookland([data[,hPos[,vPos[,anchor[,dir[,height[,narrowWidth[,humanreadable[,pen]]]]]]]]])

newBookland() creates a barcodeBooklandObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "90000"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.13.2 Methods

data()

```
data|error = <barcodeBooklandObject>:data([data])
```

data is the string containing the data for the barcode.
Bookland has two or five digits.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeBooklandObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeBooklandObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeBooklandObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()

```
height|error = <barcodeBooklandObject>:height([height])
```

height is barcode height in dots (1-999 excluding human readable) to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()
`narrowWidth|error = <barcodeBooklandObject>:narrowWidth([narrowWidth])`
narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

humanReadable()
`humanReadable|error = <barcodeBooklandObject>:humanReadable([humanReadable])`
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
`pen|error = <barcodeBooklandObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeBooklandObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeBooklandObject>:clone()`
clone() creates an exact copy of the original barcodeBooklandObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeBooklandObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.14 Barcode Interleaved 2 of 5 fields

7.3.14.1 Constructor

newInt2of5()
bcInt2of5, error = barcodeObject.newInt2of5([data[,hPos[,vPos[,anchor
[,dir[,height[,barratio[,narrowWidth[,humanreadable[,pen]]]]]]]]]])

newInt2of5() creates a barcodeInt2of5Object.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "1234567890"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
barRatio = "1:2"
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.14.2 Methods

data()

```
data|error = <barcodeInt2of5Object>:data([data])
```

data is the string containing the data for the barcode.
Interleaved 2 of 5 consists of 2 or more digits. If an odd number of digits is entered, a leading zero is added.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeInt2of5Object>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeInt2of5Object>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeInt2of5Object>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()

```
height|error = <barcodeInt2of5Object>:height([height])
```

height is barcode height in dots (1-999 excluding human readable) to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

barRatio()
```
barRatio|error = <barcodeInt2of5Object>: barRatio ([barRatio])
```
barRatio is the bar ratio, i.e. the width ratio between a thin and a wide bar, to set ["1:3"|"2:5"|"1:2"]. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode ratio is returned.

narrowWidth()
```
narrowWidth|error = <barcodeInt2of5Object>:narrowWidth([narrowWidth])
```
narrowWidth is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.
Note that if barRatio is set to "2:5", it is not possible to use an odd value of narrowWidth. If an odd value is set, the value increased by one is used.

humanReadable()
```
humanReadable|error = <barcodeInt2of5Object>:humanReadable([humanReadable])
```
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
```
pen|error = <barcodeInt2of5Object>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeInt2of5Object>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodeInt2of5Object>:clone()
```
clone() creates an exact copy of the original barcodeInt2of5Object.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeInt2of5Object>:ranges()
```
Not available on TH2.

ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.15 Barcode Ind2of5 / NEC2of5 / Code25 / Standard 2 of 5 fields

7.3.15.1 Constructor

newInd2of5()
bcInd2of5, error = barcodeObject.newInd2of5([data[,hPos[,vPos[,anchor

[,dir[,height[,barratio[,narrowWidth[,humanreadable[,pen[,sbplCompatible[,narrowSpaceSize[,wide SpaceSize[,narrowBarSize[,wideBarSize]]]]]]]]]]]]]])

newInd2of5() creates a barcodeInd2of5Object.

newNEC2of5()
bcNEC2of5, error = barcodeObject.newNEC2of5([data[,hPos[,vPos[,anchor

[,dir[,height[,barratio[,narrowWidth[,humanreadable[,pen[,sbplCompatible[,narrowSpaceSize[,wide SpaceSize[,narrowBarSize[,wideBarSize]]]]]]]]]]]]]])

newNEC2of5() creates a barcodeNEC2of5Object.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = ""
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
barRatio = "1:2"
narrowWidth = 2
narrowSpaceSize= 3
narrowBarSize = 3
wideBarSize= 5
wideSpaceSize = 5
humanReadable = true
sbplCompatible = false
pen = "NORMAL"
```

7.3.15.2 Methods

data()

```
data|error = <barcodeInd2of5Object>:data([data])
```

data is the string containing the data for the barcode.

Standard 2 of 5 consists of 2 or more digits. If an odd number of digits is entered, a leading zero is added.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()
`hPos,vPos|error = <barcodeInd2of5Object>:pos([hPos[,vPos]])`
hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()
`anchor|error = <barcodeInd2of5Object>:anchor([anchor])`
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()
`dir|error = <barcodeInd2of5Object>:dir([dir])`
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()
`height|error = <barcodeInd2of5Object>:height([height])`
height is barcode height in dots (1-999 excluding human readable) to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

barRatio()
`barRatio|error = <barcodeInd2of5Object>: barRatio ([barRatio])`
barRatio is the bar ratio, i.e. the width ratio between a thin and a wide bar, to set ["1:3"|"2:5"|"1:2"]. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode ratio is returned.

narrowWidth()
`narrowWidth|error = <barcodeInd2of5Object>:narrowWidth([narrowWidth])`
narrowWidth is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.
Note that if barRatio is set to "2:5", it is not possible to use an odd value of narrowWidth. If an odd value is set, the value increased by one is used.

humanReadable()
`humanReadable|error = <barcodeInd2of5Object>:humanReadable([humanReadable])`

humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
```
pen|error = <barcodeInd2of5Object>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeInd2of5Object>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM. NB! Gives incorrect results with humanReadable.

clone()
```
clone, error = <barcodeInd2of5Object>:clone()
```
clone() creates an exact copy of the original barcodeInd2of5Object.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeInd2of5Object>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

narrowSpaceSize()
```
narrowSpaceSize|error =
  <barcodeInd2of5Object>:narrowSpaceSize([narrowSpaceSize])
```
narrowSpaceSize is the width of a narrow space in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowSpaceSize is returned.
Note that if barRatio is set to "2:5", it is not possible to use an odd value of narrowSpaceSize. If an odd value is set, the value increased by one is used.

narrowBarSize()
```
narrowBarSize|error = <barcodeInd2of5Object>:narrowBarSize([narrowBarSize])
```
narrowBarSize is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowBarSize is returned.
Note that if barRatio is set to "2:5", it is not possible to use an odd value of narrowBarSize. If an odd value is set, the value increased by one is used.

wideSpaceSize()
```
wideSpaceSize|error = <barcodeInd2of5Object>:wideSpaceSize([wideSpaceSize])
```

wideSpaceSize is the width of a wide space in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode wideSpaceSize is returned.
Note that if barRatio is set to "2:5", it is not possible to use an odd value of wideSpaceSize. If an odd value is set, the value increased by one is used.

wideBarSize()
```
wideBarSize|error = <barcodeInd2of5Object>:wideBarSize([wideBarSize])
```
wideBarSize is the width of a wide bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode wideBarSize is returned.
Note that if barRatio is set to "2:5", it is not possible to use an odd value of wideBarSize. If an odd value is set, the value increased by one is used.

7.3.16 Barcode Code 128 fields

7.3.16.1 Constructor

newCode128()
bcCode128, error = barcodeObject.newCode128([data[,hPos[,vPos[,anchor[,dir
[,height[,narrowWidth[,humanreadable[,pen]]]]]]]]])

newCode128() creates a barcodeCode128Object.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "123ABCabc"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.16.2 Methods

data()
```
data|error = <barcodeCode128Object>:data([data])
```
data is the string containing the data for the barcode.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

Code 128 can encode ASCII values from 0 to 128.

The character ">" is used in combination with other characters to encode non-writable characters and special characters. See table below.
Note that ">J" represents the character ">".

Character combination	Subset A	Subset B	Subset C
>SPACE	NUL		
>!	SOH		
>"	STX		
>#	ETX		
>$	EOT		
>%	ENQ		
>&	ACK		
>'	BEL		
>(	BS		
>)	HT		
>*	LF		
>+	VT		
>,	FF		
>-	CR		
>.	SO		
>/	SI		
>0	DLE		
>1	DC1		
>2	DC2		
>3	DC3		
>4	DC4		
>5	NAK		
>6	SYN		
>7	ETB		
>8	CAN		
>9	EM		
>:	SUB		
>;	ESC		
><	FS		
>=	GS		
>>	RS		
>?	US	DEL	
>@	FNC3	FNC3	
>A	FNC2	FNC2	
>B	SHIFT	SHIFT	
>C	Code C	Code C	
>D	Code B	FNC4	Code B

>E	FNC4	Code A	Code A
>F	FNC1	FNC1	FNC1
>G	Start code A		
>H	Start code B		
>I	Start code C		
>J	">"		

If no start character is present, Start Code B is added automatically.
Modulo 103 check digit, and stop character are set automatically.
In subset C, an even number of digits is expected. If an odd number of digits is entered, an extra zero is added at the end of the data.

pos()
```
hPos,vPos|error = <barcodeCode128Object>:pos([hPos[,vPos]])
```
hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()
```
anchor|error = <barcodeCode128Object>:anchor([anchor])
```
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()
```
dir|error = <barcodeCode128Object>:dir([dir])
```
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()
```
height|error = <barcodeCode128Object>:height([height])
```
height is barcode height in dots (1-999 excluding human readable text and "beard") to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()
```
narrowWidth|error = <barcodeCode128Object>:narrowWidth([narrowWidth])
```
narrowWidth is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

humanReadable()
```
humanReadable|error =
<barcodeCode128Object>:humanReadable([humanReadable])
```

humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
```
pen|error = <barcodeCode128Object>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeCode128Object>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodeCode128Object>:clone()
```
clone() creates an exact copy of the original barcodeCode128Object.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeCode128Object>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.17 Barcode GS1-128 fields (Standard Carton ID only)

GS1-128 is an application standard within the Code 128 specification. It identifies data with Application Identifiers. The barcode described in this chapter sets AI to 00, i.e. Standard Carton ID. This code was previously called UCC/EAN-128.

7.3.17.1 Constructor

newGs1128()
```
bcGs1128, error = barcodeObject.newGs1128([data[,hPos[,vPos[,anchor[,dir[,height
        [,narrowWidth[,humanreadable[,humanReadableBelow[,pen]]]]]]]]])
```

newGs1128() creates a barcodeGs1-128Object.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "12345678901234567"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
humanReadable = true
humanReadableBelow = true
pen = "NORMAL"
```

7.3.17.2 Methods

data()

```
data|error = <barcodeGs1128Object>:data([data])
```

data is the string containing the data for the barcode.
GS1-128 data has seventeen digits. If less than seventeen digits are supplied, zeros will be added automatically.
Start character code (subset C), function character, Application Identification code ('00'), modulo 10 check digit, modulo 103 check digit, and stop character are set automatically.,
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeGs1128Object>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeGs1128Object>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeGs1128Object>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()

```
height|error = <barcodeGs1128Object>:height([height])
```

height is barcode height in dots (1-999 excluding human readable text and "beard") to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()
```
narrowWidth|error = <barcodeGs1128Object>:narrowWidth([narrowWidth])
```
narrowWidth is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

humanReadable()
```
humanReadable|error =
<barcodeGs1128Object>:humanReadable([humanReadable])
```
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

humanReadableBelow()
```
humanReadableBelow|error =
<barcodeGs1128Object>:humanReadablebelow([humanReadableBelow])
```
humanReadableBelow defines whether or not the interpretation line shall be printed below or abov the barcode `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadableBelow is returned.

pen()
```
pen|error = <barcodeGs1128Object>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeGs1128Object>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodeGs1128Object>:clone()
```
clone() creates an exact copy of the original barcodeGs1128Object.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeGs1128Object>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.18 Barcode GS1 Databar (RSS-14)

GS1 Databar (previously known as RSS-14) is a family of composite barcodes.

The following members of the GS1 Databar family are supported:
- GS1 Databar
- GS1 Databar Truncated
- GS1 Databar Stacked
- GS1 Databar Stacked Omnidirectional
- GS1 Databar Limited

Although their looks differ, the LUA-interface is identical (except for the name of the constructors). Hence they are all described in one chapter.

7.3.18.1 Constructors

newGs1Databar()
bcGs1Databar, error = barcodeObject.newGs1Databar([data[,hPos[,vPos[,anchor[,dir
[,narrowWidth[,pen]]]]]]])
newGs1Databar() creates a barcodeGs1DatabarObject.

newGs1DatabarT() (Truncated)
bcGs1DatabarT, error = barcodeObject.newGs1DatabarT([data[,hPos[,vPos[,anchor
[,dir[,narrowWidth[,pen]]]]]]])
newGs1DatabarT() creates a barcodeGs1DatabarTObject

newGs1DatabarS() (Stacked)
bcGs1DatabarS, error = barcodeObject.newGs1DatabarS([data[,hPos[,vPos[,anchor
[,dir[,narrowWidth[,pen]]]]]]])
newGs1DatabarS() creates a barcodeGs1DatabarSObject.

newGs1DatabarSO() (Stacked Omnidirectional)
bcGs1DatabarSO, error = barcodeObject.newGs1DatabarSO([data[,hPos[,vPos[,anchor
[,dir[,narrowWidth[,pen]]]]]]])
newGs1DatabarSO() creates a barcodeGs1DatabarSOObject.

newGs1DatabarL() (Limited)
bcGs1DatabarL, error = barcodeObject.newGs1DatabarL([data[,hPos[,vPos[,anchor
[,dir[,narrowWidth[,pen]]]]]]])
newGs1DatabarL() creates a barcodeGs1DatabarLObject.

bcGs1DatabarE, error = barcodeObject.newGs1DatabarE([data[,hPos[,vPos[,anchor
[,dir[,narrowWidth[,pen[,segmentWidth]]]]]]]])
newGs1DatabarE() creates a barcodeGs1DatabarEObject. It is only available after require("lsrender").

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.

error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "1234567890123|ABC123"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
narrowWidth = 2
pen = "NORMAL"
```

Default value data, Gs1DatabarE:
```
data = "012345678901234|ABC123"
```

7.3.18.2 Methods

data()

```
data|error = <barcodeGs1DatabarObject>:data([data])
```
data is the string containing the data for the barcode.
GS1 Databar data consist of 1-D and 2-D barcode date separated with a "|".
Up to a total of 120 characters can be entererd.
The 1-D part of the data is up to 13 digits. If less than 13 digits is provided, zeros will be added at the start to add up to 13.
The 2-dimensional part can consist of most printable characters (ASCII 32-126),
except '#', '$', '@', '[', '\', ']', '^', 'é', '{', '|', '}', and '~' (i.e. ASCII 35, 36, 64, 91-94, 96, and 123-127).
A check digit is added automatically.

pos()
```
hPos,vPos|error = <barcodeGs1DatabarObject>:pos([hPos[,vPos]])
```
hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()
```
anchor|error = <barcodeGs1DatabarObject>:anchor([anchor])
```
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()
```
dir|error = <barcodeGs1DatabarObject>:dir([dir])
```
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

narrowWidth()
```
narrowWidth|error = <barcodeGs1DatabarObject>:narrowWidth([narrowWidth])
```

narrowWidth is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.
Note that the height of the barcode is set in proportion to the chosen narrowWidth.

pen()
`pen|error = <barcodeGs1DatabarObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeGs1DatabarObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM. NB! barcodeGs1DatabarEObject gives incorrect results.

clone()
`clone, error = <barcodeGs1DatabarObject>:clone()`
clone() creates an exact copy of the original barcodeGs1DatabarObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeGs1DatabarObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

segmentWidth()
`segmentWidth|error= <barcodeGs1DatabarEObject>:segmentWidth([segmentWidth])`
In the Gs1DatabarE barcode, it is possible to specify the segmentWidth. It is an even number.

7.3.19 Barcode EAN-8 Composite fields

Barcode EAN-8 Composite is a normal EAN-8 barcode with a composite field added on top of it.

7.3.19.1 Constructor

newEan8C()
bcEan8C, error = barcodeObject.newEan8C([data[,hPos[,vPos[,anchor[,dir
[,narrowWidth[,pen]]]]]]])

newEan8C() creates a barcodeEan8CObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.

error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "12345670|ABC123"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
narrowWidth = 2
pen = "NORMAL"
```

7.3.19.2 Methods

data()

```
data|error = <barcodeEan8CObject>:data([data])
```

data is the string containing the data for the barcode.
The data consists of a 1-dimensional (linear) and a 2-dimensional part. These parts are separated with the character "|" (ASCII 124).

The 1-dimensional part consists of seven digits and a check digit. If the string contains seven digits, a check digit is generated and added. If the string contains eight digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.

The 2-dimensional part can consist of most printable characters (ASCII 32-126),
except '#', '$', '@', '[', '\', ']', '^', 'é', '{', '|', '}', and '~' (i.e. ASCII 35, 36, 64, 91-94, 96, and 123-127).

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeEan8CObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeEan8CObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeEan8CObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

narrowWidth()
`narrowWidth|error = <barcodeEan8CObject>:narrowWidth([narrowWidth])`
narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.
Note that the height of the barcode is set in proportion to the chosen narrowWidth.

pen()
`pen|error = <barcodeEan8CObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeEan8CObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeEan8CObject>:clone()`
clone() creates an exact copy of the original barcodeEan8CObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeEan8CObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.20 Barcode JAN-8 Composite fields

Barcode JAN-8 Composite is a normal JAN-8 barcode with a composite field added on top of it. To avoid any compatibility problems, our implementation of JAN-8 Composite will allow all valid EAN-8 Composite data. This means that the 1-dimensional part does not have to start with “45” or “49”.

7.3.20.1 Constructor

newJan8C()
bcJan8C, error = barcodeObject.newJan8C([data[,hPos[,vPos[,anchor[,dir
[,narrowWidth[,pen]]]]]]])

newJan8C() creates a barcodeJan8CObject.

If all parameters are left out the default for each parameter will be used (see below).

Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "49012347|ABC123"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
narrowWidth = 2
pen = "NORMAL"
```

7.3.20.2 Methods

data()

```
data|error = <barcodeJan8CObject>:data([data])
```

data is the string containing the data for the barcode.
The data consists of a 1-dimensional (linear) and a 2-dimensional part. These parts are separated with the character "|" (ASCII 124).

The 1-dimensional part consists of seven digits and a check digit. If the string contains seven digits, a check digit is generated and added. If the string contains eight digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.

The 2-dimensional part can consist of most printable characters (ASCII 32-126), except '#', '$', '@', '[', '\', ']', '^', 'é', '{', '|', '}', and '~' (i.e. ASCII 35, 36, 64, 91-94, 96, and 123-127).

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeJan8CObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeJan8CObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeJan8CObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

narrowWidth()
`narrowWidth|error = <barcodeJan8CObject>:narrowWidth([narrowWidth])`
narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.
Note that the height of the barcode is set in proportion to the chosen narrowWidth.

pen()
`pen|error = <barcodeJan8CObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeJan8CObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeJan8CObject>:clone()`
clone() creates an exact copy of the original barcodeJan8CObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeJan8CObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.21 Barcode EAN-13 Composite fields

Barcode EAN-13 Composite is a normal EAN-13 barcode with a composite field added on top of it.

7.3.21.1 Constructor

newEan13C()
bcEan13C, error = barcodeObject.newEan13C([data[,hPos[,vPos[,anchor[,dir
[,narrowWidth[,pen]]]]]]])

newEan13C() creates a barcodeEan13CObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "1234567890128|ABC123"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
narrowWidth = 2
pen = "NORMAL"
```

7.3.21.2 Methods

data()

```
data|error = <barcodeEan13CObject>:data([data])
```

data is the string containing the data for the barcode.
The data consists of a 1-dimensional (linear) and a 2-dimensional part. These parts are separated with the character "|" (ASCII 124).

The 1-dimensional part consists of twelve digits and a check digit. If the string contains twelve digits, a check digit is generated and added. If the string contains thirteen digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.

The 2-dimensional part can consist of most printable characters (ASCII 32-126), except '#', '$', '@', '[', '\', ']', '^', 'é', '{', '|', '}', and '~' (i.e. ASCII 35, 36, 64, 91-94, 96, and 123-127).

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeEan13CObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeEan13CObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeEan13CObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

narrowWidth()

```
narrowWidth|error = <barcodeEan13CObject>:narrowWidth([narrowWidth])
```
narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.
Note that the height of the barcode is set in proportion to the chosen narrowWidth.

pen()
```
pen|error = <barcodeEan13CObject>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeEan13CObject>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodeEan13CObject>:clone()
```
clone() creates an exact copy of the original barcodeEan13CObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeEan13CObject>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.22 Barcode JAN-13 Composite fields

Barcode JAN-13 Composite is a normal JAN-13 barcode with a composite field added on top of it. To avoid any compatibility problems, our implementation of JAN-13 Composite will allow all valid EAN-13 Composite data. This means that the 1-dimensional part does not have to start with “45” or “49”.

7.3.22.1 Constructor

newJan13C()
```
bcJan13C, error = barcodeObject.newJan13C([data[,hPos[,vPos[,anchor[,dir
                                [,narrowWidth[,pen]]]]]]])
```

newJan13C() creates a barcodeJan13CObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.

error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "4901234567894|ABC123"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
narrowWidth = 2
pen = "NORMAL"
```

7.3.22.2 Methods

data()

```
data|error = <barcodeJan13CObject>:data([data])
```

data is the string containing the data for the barcode.
The data consists of a 1-dimensional (linear) and a 2-dimensional part. These parts are separated with the character "|" (ASCII 124).

The 1-dimensional part consists of twelve digits and a check digit. If the string contains twelve digits, a check digit is generated and added. If the string contains thirteen digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.

The 2-dimensional part can consist of most printable characters (ASCII 32-126), except '#', '$', '@', '[', '\', ']', '^', 'é', '{', '|', '}', and '~' (i.e. ASCII 35, 36, 64, 91-94, 96, and 123-127).

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeJan13CObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeJan13CObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeJan13CObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

narrowWidth()
`narrowWidth|error = <barcodeJan13CObject>:narrowWidth([narrowWidth])`
narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.
Note that the height of the barcode is set in proportion to the chosen narrowWidth.

pen()
`pen|error = <barcodejan13CObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeJan13CObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeJan13CObject>:clone()`
clone() creates an exact copy of the original barcodeJan13CObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeJan13CObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.23 Barcode UPC-A Composite fields

Barcode UPC-A Composite is a normal UPC-A barcode with a composite field added on top of it.

7.3.23.1 Constructor

newUpcaC()
bcUpcaC, error = barcodeObject.newUpcaC([data[,hPos[,vPos[,anchor[,dir
[,narrowWidth[,pen]]]]]]])

newUpcaC() creates a barcodeUpcaCObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "12345678901|ABC123"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
narrowWidth = 2
pen = "NORMAL"
```

7.3.23.2 Methods

data()

```
data|error = <barcodeUpcaCObject>:data([data])
```

data is the string containing the data for the barcode.
The data consists of a 1-dimensional (linear) and a 2-dimensional part. These parts are separated with the character "|" (ASCII 124).

The 1-dimensional part consists of eleven digits and a check digit. If the string contains eleven digits, a check digit is generated and added. If the string contains twelve digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.

The 2-dimensional part can consist of most printable characters (ASCII 32-126), except '#', '$', '@', '[', '\', ']', '^', 'é', '{', '|', '}', and '~' (i.e. ASCII 35, 36, 64, 91-94, 96, and 123-127).

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeUpcaCObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeUpcaCObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeUpcaCObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

narrowWidth()

```
narrowWidth|error = <barcodeUpcaCObject>:narrowWidth([narrowWidth])
```

narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.
Note that the height of the barcode is set in proportion to the chosen narrowWidth.

pen()
`pen|error = <barcodeUpcaCObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeUpcaCObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeUpcaCObject>:clone()`
clone() creates an exact copy of the original barcodeUpcaCObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeUpcaCObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.24 Barcode UPC-E Composite fields

Barcode UPC-E Composite is a normal UPC-E barcode with a composite field added on top of it.

7.3.24.1 Constructor

newUpceC()
bcUpceC, error = barcodeObject.newUpceC([data[,hPos[,vPos[,anchor[,dir
[,narrowWidth[,pen]]]]]]])

newUpceC() creates a barcodeUpceCObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
`data = "123456|ABC123"`

```
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
narrowWidth = 2
pen = "NORMAL"
```

7.3.24.2 Methods

data()

```
data|error = <barcodeUpceCObject>:data([data])
```

data is the string containing the data for the barcode.
The data consists of a 1-dimensional (linear) and a 2-dimensional part. These parts are separated with the character "|" (ASCII 124).

The 1-dimensional part consists of six digits and a check digit. If the string contains six digits, a check digit is generated and added. If the string contains seven digits, the provided check digit is verified. If the check digit is wrong, the new data is not accepted.

The 2-dimensional part can consist of most printable characters (ASCII 32-126),
except '#', '$', '@', '[', '\', ']', '^', 'é', '{', '|', '}', and '~' (i.e. ASCII 35, 36, 64, 91-94, 96, and 123-127).

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeUpceCObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeUpceCObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeUpceCObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

narrowWidth()

```
narrowWidth|error = <barcodeUpceCObject>:narrowWidth([narrowWidth])
```

narrowWidth is the width of a narrow bar in dots to set (2-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.
Note that the height of the barcode is set in proportion to the chosen narrowWidth.

pen()
`pen|error = <barcodeUpceCObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeUpceCObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeUpceCObject>:clone()`
clone() creates an exact copy of the original barcodeUpceCObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeUpceCObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.25 Barcode GS1 128 Composite CC-A/B fields

Barcode GS1 128 Composite CC-A/B is a normal GS1 128 barcode with a CC-A/B composite field added on top of it.

7.3.25.1 Constructor

newGs1128Cab()
bcGs1128Cab, error = barcodeObject.newGs1128Cab([data[,hPos[,vPos[,anchor
[,dir[,height[,narrowWidth[,pen]]]]]]]])

newGs1128Cab() creates a barcodeGs1128CabObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "1234567890|ABC123"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
pen = "NORMAL"
```

7.3.25.2 Methods

data()

```
data|error = <barcodeGs1128CabObject>:data([data])
```

data is the string containing the data for the barcode.
The data consists of a 1-dimensional (linear) and a 2-dimensional part. These parts are separated with the character "|" (ASCII 124).

The 1-dimensional part consists of 5 (3 if at least one character is none digit) up to 48 characters. A leading FNC1 char ('#', ASCII 35) will be added automatically if not provided.

Both the 1 and the 2-dimensional part can consist of most printable characters (ASCII 32-126), except '#', '$', '@', '[', '\', ']', '^', 'é', '{', '|', '}', and '~' (i.e. ASCII 35, 36, 64, 91-94, 96, and 123-127).

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeGs1128CabObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeGs1128CabObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeGs1128CabObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()

```
height|error = <barcodeGs1128CabObject>:height([height])
```

height is barcode height in dots to set (1-999), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()
```
narrowWidth|error = <barcodeGs1128CabObject>:narrowWidth([narrowWidth])
```
narrowWidth is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

pen()
```
pen|error = <barcodeGs1128CabObject>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeGs1128CabObject>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodeGs1128CabObject>:clone()
```
clone() creates an exact copy of the original barcodeGs1128CabObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeGs1128CabObject>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.26 Barcode GS1 128 Composite CC-C fields

Barcode GS1 128 Composite CC-C is a normal GS1 128 barcode with a CC-C composite field added on top of it.

7.3.26.1 Constructor

newGs1128Cc()
bcGs1128Cc, error = barcodeObject.newGs1128Cc([data[,hPos[,vPos[,anchor[,dir
[,height[,narrowWidth[,pen]]]]]]]])

newGs1128Cc() creates a barcodeGs1128CcObject.

If all parameters are left out the default for each parameter will be used (see below).

Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "1234567890|ABC123"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
pen = "NORMAL"
```

7.3.26.2 Methods

data()

```
data|error = <barcodeGs1128CcObject>:data([data])
```

data is the string containing the data for the barcode.
The data consists of a 1-dimensional (linear) and a 2-dimensional part. These parts are separated with the character "|" (ASCII 124).

The 1-dimensional part consists of 5 (3 if at least one character is none digit) up to 48 characters. A leading FNC1 char ('#', ASCII 35) will be added automatically if not provided.

Both the 1 and the 2-dimensional part can consist of most printable characters (ASCII 32-126), except '#', '$', '@', '[', '\', ']', '^', 'é', '{', '|', '}', and '~' (i.e. ASCII 35, 36, 64, 91-94, 96, and 123-127).

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeGs1128CcObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeGs1128CcObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeGs1128CcObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()
`height|error = <barcodeGs1128CcObject>:height([height])`
height is barcode height in dots to set (1-999), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()
`narrowWidth|error = <barcodeGs1128CcObject>:narrowWidth([narrowWidth])`
narrowWidth is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

pen()
`pen|error = <barcodeGs1128CcObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeGs1128CcObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeGs1128CcObject>:clone()`
clone() creates an exact copy of the original barcodeGs1128CcObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeGs1128CcObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.27 Barcode MSI fields

The MSI barcode, or Modified Plessey, is a barcode encoding digits 0-9.

7.3.27.1 Constructor

newMSI()
bcMSI, error = barcodeObject.newMSI([data[,hPos[,vPos[,anchor
[,dir[,height[,narrowWidth[,humanreadable[,pen]]]]]]]]])

newMSI() creates a barcodeMSIObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = " 1234567890123"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
height = 50
narrowWidth = 2
humanReadable = true
pen = "NORMAL"
```

7.3.27.2 Methods

data()

```
data|error = <barcodeMSIObject>:data([data])
```

data is the string containing the data for the barcode.
Interleaved 2 of 5 consists of 2 or more digits. If an odd number of digits is entered, a leading zero is added.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeMSIObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeMSIObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeMSIObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

height()

```
height|error = <barcodeMSIObject>:height([height])
```

height is barcode height in dots (1-999 excluding human readable) to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode height is returned.

narrowWidth()

```
narrowWidth|error = <barcodeMSIObject>:narrowWidth([narrowWidth])
```

narrowWidth is the width of a narrow bar in dots to set (1-12), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current barcode narrowWidth is returned.

Note that if barRatio is set to "2:5", it is not possible to use an odd value of narrowWidth. If an odd value is set, the value increased by one is used.

humanReadable()

```
humanReadable|error = <barcodeMSIObject>:humanReadable([humanReadable])
```

humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()

```
pen|error = <barcodeMSIObject>:pen([pen])
```

pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()

```
hPos, vPos, width, height, error = <barcodeMSIObject>:fieldSize()
```

Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM. NB! Gives incorrect results with humanReadable in particular.

clone()

```
clone, error = <barcodeMSIObject>:clone()
```

clone() creates an exact copy of the original barcodeMSIObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()

```
tbl = <barcodeMSIObject>:ranges()
```

Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.28 Barcode Customer fields

This is available after require("lsrender").

Customer is a barcode that is used by POST services. It is also known as RM4SCC. The data is divided into two sections: <Postal Code, 7 digits><Print Data, up to 13 bytes>. The <Print Data>-section can contain a hyphen (-), 0-9 and letters A-Z.

7.3.28.1 Constructor

newCustomer()
bcCUSTOMER, error = barcodeObject.newCustomer([data[,hPos[,vPos[,anchor [,dir [,humanreadable[,pen]]]]]]])

newCUSTOMER() creates a barcodeCustomerObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "1234567,1-12345678901"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
humanReadable = true
pen = "NORMAL"
```

7.3.28.2 Methods

data()

```
data|error = <barcodeCustomerObject>:data([data])
```

data is the string containing the data for the barcode.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeCustomerObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeCustomerObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeCustomerObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

humanReadable()
`humanReadable|error = <barcodeCustomerObject>:humanReadable([humanReadable])`
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
`pen|error = <barcodeCustomerObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeCustomerObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM. NB! barcodeCustomerObject gives incorrect results with humanReadable.

clone()
`clone, error = <barcodeCustomerObject>:clone()`
clone() creates an exact copy of the original barcodeCustomerObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeCustomerObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.29 Barcode POSTNET / USPS fields

These are available after require("lsrender").
POSTNET is a barcode that was used by the US POST. The number of digits in that data field, specified the format: 5 – POSTNET-32, 6 – POSTNET-37, 9 – POSTNET-52, 11 – POSTNET-62 Delivery Point.

USPS is a barcode associated with the United States Postal Service. The digits in the data field make up the <Barcode ID,2 digits><Service type ID,3 digits><Mailer ID,6 digits><Serial number, 9 digits>[<Routing code,5 or 9 or 11 digits>]. The <Routing code> can be omitted.

7.3.29.1 Constructor

newPOSTNET()
bcPOSTNET, error = barcodeObject.newPOSTNET([data[,hPos[,vPos[,anchor

[,dir [,humanreadable[,pen]]]]]]])

newPOSTNET() creates a barcodePOSTNETObject.

newUSPS()
bcUSPS, error = barcodeObject.newUSPS([data[,hPos[,vPos[,anchor
[,dir [,humanreadable[,pen]]]]]]])

newUSPS() creates a barcodeUSPSObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "12345"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
humanReadable = true
pen = "NORMAL"
```

Default value (USPS):
```
data = "53379777234994544928511357594 61"
```

7.3.29.2 Methods

data()

```
data|error = <barcodePOSTNETObject>:data([data])
```

data is the string containing the data for the barcode.
error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodePOSTNETObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodePOSTNETObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()
`dir|error = <barcodePOSTNETObject>:dir([dir])`
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

humanReadable()
`humanReadable|error = <barcodePOSTNETObject>:humanReadable([humanReadable])`
humanReadable defines whether or not an interpretation line shall be printed `[true|false]`. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current value of humanReadable is returned.

pen()
`pen|error = <barcodePOSTNETObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodePOSTNETObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM. NB! Gives incorrect results.

clone()
`clone, error = <barcodePOSTNETObject>:clone()`
clone() creates an exact copy of the original barcodePOSTNETObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodePOSTNETObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.30 Barcode Aztec fields

7.3.30.1 Constructor

newAztec()
bcAztec, error = barcodeObject.newAztec([data[,hPos[,vPos[,anchor

[,dir[,hSize[,vSize[,compact[,securityLevel[,cellSize[,encoding[,messageData[,message[,pen]]]]]]]]]]]])

newAztec() creates a barcodeAztecObject. It is available after require("lsrender").

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "123456ABCDEF"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
cellSize = 2
securityLevel = 0
messageData = "DEFAULT VALUE"
message = false
encoding = 0
compact = true
hSize = 0
vSize = 0
pen = "NORMAL"
```

7.3.30.2 Methods

data()

```
data|error = <barcodeAztecObject>:data([data])
```

data is the string containing the data for the barcode.

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

message()

```
message|error = <barcodeAztecObject>:message([message])
```

message is a number for the barcode.

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

messageData()

```
messageData|error = <barcodeAztecObject>:messageData([messageData])
```

messageData is the messageData for the barcode.

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

securityLevel()

```
securityLevel|error = <barcodeAztecObject>:securityLevel([securityLevel])
```

securityLevel controls the securityLevel for the barcode. It can be specified between 0 and 99.

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

encoding()

```
encoding|error = <barcodeAztecObject>:encoding([encoding])
```

encoding controls the encoding for the barcode. It can be specified between 0 and 26.

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

compact()

```
compact|error = <barcodeAztecObject>:compact([compact])
```

compact controls the symbol type for the barcode. When true, it means it uses the compact symbol.

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeAztecObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeAztecObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeAztecObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

cellSize()

```
cellSize|error = <barcodeAztecObject>:cellSize([cellSize])
```

cellSize sets the width and height in dots (0-32) of the square shaped dots that builds the barcode. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current cellSize is returned.

pen()

```
pen|error = <barcodeAztecObject>:pen([pen])
```

pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeAztecObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM. NB! Gives incorrect values.

clone()
`clone, error = <barcodeAztecObject>:clone()`
clone() creates an exact copy of the original barcodeAztecObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

7.3.31 Barcode Data Matrix fields

7.3.31.1 Constructor

newDatamatrix()
bcDatamatrix, error = barcodeObject.newDatamatrix([data[,hPos[,vPos[,anchor
[,dir[,cellSize[,hSize[,vSize[,pen]]]]]]]]])

newDatamatrix() creates a barcodeDatamatrixObject.

newGs1Datamatrix()
bcGs1Datamatrix, error = barcodeObject.newGs1Datamatrix([data[,hPos[,vPos[,anchor
[,dir[,cellSize[,hSize[,vSize[,pen]]]]]]]]])

newGs1Datamatrix() creates a barcodeGS1DatamatrixObject. It is available after require("lsrender").

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "123456ABCDEF"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
cellSize = 2
hSize = 0
vSize = 0
pen = "NORMAL"
```

7.3.31.2 Methods

data()

```
data|error = <barcodeDatamatrixObject>:data([data])
```

data is the string containing the data for the barcode.
The data can include all values between 0 and 255. Values without corresponding printable character are input with the standard Lua syntax, i.e. '\<decimal value>.
Example: Two ways of setting data to "Hello world!"
- bcDatamatrix:data("Hello world!")
- bcDatamatrix:data("\072\101\108\108\111\032\119\111\114\108\100\033")

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeDatamatrixObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeDatamatrixObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeDatamatrixObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

cellSize()

```
cellSize|error = <barcodeDatamatrixObject>:cellSize([cellSize])
```

cellSize sets the width and height in dots (1-16) of the square shaped dots that builds the barcode. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current cellSize is returned.

size()

```
hSize,vSize|error = <barcodeDatamatrixObject>:size([hSize[,vSize]])
```

hSize, vSize are number of cells horizontally and vertically. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hSize and vSize are returned.

The default values (0, 0) mean AUTO. In his case the smallest possible square shaped barcode will be rendered.

Note that the standard only allows some combinations of hSize x vSize (10x10,12x12, 14x14, 16x16, 18x18, 20x20, 22x22, 24x24, 26x26, 32x32, 36x36, 40x40, 44x44, 48x48, 52x52, 64x64, 72x72, 80x80, 88x88, 96x96, 104x104, 120x120, 132x132, 144x144, 18x8, 32x8, 26x12, 36x12, 36x16, and 48x16).

pen()
`pen|error = <barcodeDatamatrixObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeDatamatrixObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeDatamatrixObject>:clone()`
clone() creates an exact copy of the original barcodeDatamatrixObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeDatamatrixObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.32 Barcode QR Code fields

7.3.32.1 Constructor

newQrcode()
bcQrcode, error = barcodeObject.newQrcode([data[,hPos[,vPos[,anchor[,dir,
[correctionLevel[,cellSize[,encoding[,pen]]]]]]]]])

newQrcode() creates a barcodeQrcodeObject.

newQrcodeModel1()
bcQrcodemodel1, error = barcodeObject.newQrcode([data[,hPos[,vPos[,anchor[,dir,
[correctionLevel[,cellSize[,encoding[,pen]]]]]]]]])

newQrcodeModel1() creates a barcodeQrcodeModel1Object.

newSQrcode ()
bcSQrcodemodel, error = barcodeObject.newSQrcode([data[,hPos[,vPos[,anchor[,dir,
[correctionLevel[,cellSize[,encoding[,pen[,keyData[,secureData]]]]]]]]]])

newSQrcode () creates a barcodeSQrcodeObject.

newQrcodeModel1(), newSQrcode() are enabled with require("lsrender"), and the newQrcode() implementation is also controlled by require("lsrender").

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "123456ABCDEF"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
correctionlevel = "M"
cellSize = 4
encoding = 4
pen = "NORMAL"
```

Default values SQrcode:

```
sqrKeyData = "0000000000000000"
sqrData = "123456ABCDEF"
```

7.3.32.2 Methods

data()

```
data|error = <barcodeQrcodeObject>:data([data])
```

data is the string containing the data for the barcode.
The data can include all values between 0 and 255. Values without corresponding printable character are input with the standard Lua syntax, i.e. '\<decimal value>.
Example: Two ways of setting data to "Hello world!"
- bcQrcode:data("Hello world!")
- bcQrcode:data("\072\101\108\108\111\032\119\111\114\108\100\033")

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeQrcodeObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeQrcodeObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeQrcodeObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

correctionLevel()

```
correctionLevel|error = <barcodeQrcodeObject>:correctionLevel([correctionLevel])
```

correctionLevel sets the level of redundancy in the code. It can be set to "L" (high density, 7%), "M" (standard, 15%), "Q" (high reliability, 25%), or "H" (ultra high reliability, 30%).
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current correctionLevel is returned.

cellSize()

```
cellSize|error = <barcodeQrcodeObject>:cellSize([cellSize])
```

cellSize sets the width and height in dots (1-32) of the square shaped dots that builds the barcode.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current cellSize is returned.

encoding()

```
encoding|error = <barcodeQrcodeObject>:encoding([encoding])
```

encoding sets the character mode. it can be set to 0, 1, 2, 3, or 4.
0 – Numeric
Allowed characters: '0' to '9' (ASCII 48-57)
1 – Alphanumeric
Allowed characters: '0' to '9', 'A' to 'Z', ':', ' ', '$', '%', '*', '+', '-', '.', and '/' (ASCII 32, 36, 37, 42, 43, 45-58, and 65-90)
2 – Binary
Allowed characters: ASCII 0-127 and 160-223.
3 – Kanji
16 bit characters according to the Shift JIS system.
Allowed characters: 8000(hex)-9FFF(hex) and E000(hex)-FFFF(hex)
4 – Auto
Allowed characters: ASCII 0-255

error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current cellSize is returned.

sqrKeyData()
`sqrKeyData|error = <barcodeSQrcodeObject>:sqrKeyData([sqrKeyData])`
sqrKeyData is the Secure Qrcode KeyData. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current sqrKeyData is returned.

sqrData()
`sqrData|error = <barcodeSQrcodeObject>:sqrData([sqrData])`
sqrData is the Secure Qrcode Data. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current sqrData is returned.

pen()
`pen|error = <barcodeQrcodeObject>:pen([pen])`
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
`hPos, vPos, width, height, error = <barcodeQrcodeObject>:fieldSize()`
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <barcodeQrcodeObject>:clone()`
clone() creates an exact copy of the original barcodeQrcodeObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <barcodeQrcodeObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.33 Barcode Micro QR Code fields

7.3.33.1 Constructor

newMicroQrcode()
bcMicroQrcode, error = barcodeObject.newMicroQrcode([data[,hPos[,vPos[,anchor
[,dir,[correctionLevel[,cellSize[,encoding[,pen]]]]]]]]])

newMicroQrcode() creates a barcodeMicroQrcodeObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
data = "123456ABCDEF"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
correctionlevel = "M"
cellSize = 4
encoding = 4
pen = "NORMAL"
```

7.3.33.2 Methods

data()

```
data|error = <barcodeMicroQrcodeObject>:data([data])
```

data is the string containing the data for the barcode.
The data can include all values between 0 and 255. Values without corresponding printable character are input with the standard Lua syntax, i.e. '\<decimal value>.
Example: Two ways of setting data to "Hello world!"
- bcMicroQrcode:data("Hello world!")
- bcMicroQrcode:data("\072\101\108\108\111\032\119\111\114\108\100\033")

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()

```
hPos,vPos|error = <barcodeMicroQrcodeObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()

```
anchor|error = <barcodeMicroQrcodeObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()

```
dir|error = <barcodeMicroQrcodeObject>:dir([dir])
```

dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

correctionLevel()
```
correctionLevel|error =
   <barcodeMicroQrcodeObject>:correctionLevel([correctionLevel])
```
correctionLevel sets the level of redundancy in the code. It can be set to "L" (high density, 7%), "M" (standard, 15%), or "Q" (high reliability, 25%). Note that correctionLevel "H" is not available for Micro QR Code..
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current correctionLevel is returned.

cellSize()
```
cellSize|error = <barcodeMicroQrcodeObject>:cellSize([cellSize])
```
cellSize sets the width and height in dots (1-32) of the square shaped dots that builds the barcode.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current cellSize is returned.

encoding()
```
encoding|error = <barcodeMicroQrcodeObject>:encoding([encoding])
```
encoding sets the character mode. it can be set to 0, 1, 2, 3, or 4.
0 – Numeric
Allowed characters: '0' to '9' (ASCII 48-57)
1 – Alphanumeric
Allowed characters: '0' to '9', 'A' to 'Z', ':', ' ', '$', '%', '*', '+', '-', '.', and '/' (ASCII 32, 36, 37, 42, 43, 45-58, and 65-90)
2 – Binary
Allowed characters: ASCII 0-127 and 160-223.
3 – Kanji
16 bit characters according to the Shift JIS system.
Allowed characters: 8000(hex)-9FFF(hex) and E000(hex)-FFFF(hex)
4 – Auto
Allowed characters: ASCII 0-255
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameter current cellSize is returned.

pen()
```
pen|error = <barcodeMicrQrcodeObject>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeMicroQrcodeObject>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodeMicroQrcodeObject>:clone()
```
clone() creates an exact copy of the original barcodeMicroQrcodeObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeMicroQrcodeObject>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.34 Barcode MaxiCode fields

7.3.34.1 Constructor

newMaxicode()
bcMaxicode, error = barcodeObject.newMaxicode([data[,hPos[,vPos[,anchor[,dir
[,number[,maxNumber[,pen]]]]]]]])
newMaxicode() creates a barcodeMaxicodeObject.

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "4123ABC"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
number = 1
maxNumber = 1
pen = "NORMAL"
```

7.3.34.2 Methods

data()

```
data|error = <barcodeMaxicodeObject>:data([data])
```
data is the string containing the data for the barcode.

The data can include all values between 0 and 255. The maximum number of characters depends on the mix of data and range from 93 (no digits) to 138 (only digits).
Values without corresponding printable character are input with the standard Lua syntax, i.e. '\<decimal value>.
Example: Two ways of setting data to "Hello!"
- bcMaxicode:data("4Hello!")

- bcMaxicode:data("\052\072\101\108\108\111\033")

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

The first character in the data string defines the mode. In the example above, mode is set to 4.
Four modes are supported:
Mode 2
Formatted data containing a structured Carrier Message with a numeric postal code.
Primary use is US domestic destinations.
Mode 3
Formatted data containing a structured Carrier Message with an alphanumeric postal code. Primary use is international destinations.
Mode 4
Unformatted data with Standard Error Correction.
Mode 6
Used for programming hardware devices.

For mode 2 and 3, there is a strict syntax for the data following the mode character.
<mode> <class> <country zip> <zip> <message>
<mode>
“2” or “3” depending on mode (as mentioned above).
<class>
Three digit defining service class
<country zip>
Three digit country zip code.
<zip>
Zip code.
Mode 2 - Nine digit zip code. Pad with trailing zeros if only five is used.
Mode 3 – Six character alphanumeric zip code.
<message>
The following fields are included in the message part:
Header, “[)><RS>01<GS>96”
Tracking number, “<10 or 11 alphanumeric chars><GS>”
SCAC, “UPSN<GS>”
UPS shipper number, “<6 chars><GS>”
Day of year of pickup (1-366), “<3 digits><GS>”
Shipment id number, “<0-30 chars><GS>”
Package X of Y, <1-3 digits>/<1-3 digits>“<GS>”
Package weight, “<1-3 digits><GS>”
Address validation,<”Y” or “N”> “<GS>”
Ship to address, “<0-35 chars><GS>”
Ship to city, “<1-20 chars><GS>”
Ship to state, “<2 chars>”

End of message, “<RS><EOT>”

<RS>: ASCII Record Separator, decimal 30 (\030).
<GS>: ASCII Group Separator, decimal 29 (\029).
<EOT>: ASCII End of transmission, decimal 4 (\004).

Example:
bcMaxicode:data("2001840282730000[)>\03001\029961Z00004952\029UPSN\0299BCJ50\029365\0292011110701\0291/1\02910\029Y\02910350A NF RD\029CHARLOTTE\029NC\030\004")

pos()
```
hPos,vPos|error = <barcodeMaxicodeObject>:pos([hPos[,vPos]])
```
hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()
```
anchor|error = <barcodeMaxicodeObject>:anchor([anchor])
```
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()
```
dir|error = <barcodeMaxicodeObject>:dir([dir])
```
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

series()
```
number,maxnumber|error = <barcodeMaxicodeObject>:series([number[,maxNumber]])
```
Up to eight maxiCode labels can be used in a series for one shipment. number is the label number for the current label. maxNumber is the total number of labels in the series.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current number and maxNumber are returned.

pen()
```
pen|error = <barcodeMaxicodeObject>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodeMaxicodeObject>:fieldSize()
```

Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodeMaxicodeObject>:clone()
```
clone() creates an exact copy of the original barcodeMaxicodeObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodeMaxicodeObject>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.35 Barcode PDF417, MicroPDF417 fields

7.3.35.1 Constructor

newPdf417()
bcPdf417, error = barcodeObject.newPdf417(data[,hPos[,vPos[,anchor[,dir
[,cellWidth[,cellHeight[,rows[,columns[securityLevel[,truncate[,pen]]]]]]]]]])

newPdf417() creates a barcodePdf417Object.

newMicroPdf417()
bcMicroPdf417, error = barcodeObject.newMicroPdf417(data[,hPos[,vPos[,anchor[,dir
[,cellWidth[,cellHeight[,rows[,columns[securityLevel[,truncate[,pen[,binaryMode]]]]]]]]]])

newMicroPdf417() creates a barcodeMicroPdf417Object. It is available after require("lsrender").

If all parameters are left out the default for each parameter will be used (see below).
Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
data = "123ABC"
hPos = 100
vPos = 100
anchor = "TOP_LEFT"
dir = 0
cellWidth = 3
cellHeight = 9
rows = 0
columns = 0
securityLevel = 0
```

```
truncate = false
pen = "NORMAL"
```

Default value (MicroPdf417):
```
binaryMode = false
```

7.3.35.2 Methods

data()

```
data|error = <barcodePdf417Object>:data([data])
```
data is the string containing the data for the barcode.

The data can include up to 2710 digits or 1850 text characters or 1108 binary bytes. Values without corresponding printable character are input with the standard Lua syntax, i.e. '\<decimal value>.
Example: Two ways of setting data to "Hello!"
- bcPdf417:data("Hello!")
- bcPdf417:data("\072\101\108\108\111\033")

error is set to errno.ESUCCESS if ok, otherwise errno.EPARAM. If called without parameter, current data is returned.

pos()
```
hPos,vPos|error = <barcodePdf417Object>:pos([hPos[,vPos]])
```
hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hPos and vPos are returned.

anchor()
```
anchor|error = <barcodePdf417Object>:anchor([anchor])
```
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

dir()
```
dir|error = <barcodePdf417Object>:dir([dir])
```
dir is text printing direction `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

cellWidth()
```
cellWidth|error = <barcodePdf417Object>:cellWidth([cellWidth])
```
cellWidth sets the width in dots (1-9) of the “pixels” that builds the barcode.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current cellWidth is returned.

cellHeight()
```
cellHeight|error = <barcodePdf417Object>:cellHeight([cellHeight])
```

cellHeight sets the height in dots (1-24) of the “pixels” that builds the barcode.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current cellHeight is returned.

rows()
```
rows|error = <barcodePdf417Object>:rows([rows])
```
rows sets the number of data rows (3-90) in the barcode. If set to 0 (default), the number of rows is set automatically. If both rows and columns are set (to non-zero) and the data does not fit, then the barcode will not be printed.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current rows is returned.

columns()
```
columns|error = <barcodePdf417Object>:columns([columns])
```
columns sets the number of data columns (1-30) in the barcode. If set to 0 (default), the number of columns is set automatically. If both rows and columns are set (to non-zero) and the data does not fit, then the barcode will not be rendered.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current columns is returned.

securityLevel()
```
securityLevel|error = <barcodePdf417Object>:securityLevel([securityLevel])
```
securityLevel (0-8) sets the level of error correction for the barcode. If set to 0 (default), then only detection is available. Increased securityLevel will increase the level of error correction as well as the size of the barcode.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current securityLevel is returned.

truncate()
```
truncate|error = <barcodePdf417Object>:truncate([truncate])
```
truncate `[true|false]` sets whether a normal (false) or a compact/truncated version of the barcode should be printed.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

pen()
```
pen|error = <barcodePdf417Object>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <barcodePdf417Object>:fieldSize()
```

Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <barcodePdf417Object>:clone()
```
clone() creates an exact copy of the original barcodePdf417Object.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <barcodePdf417Object>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

binaryMode()
```
binaryMode|error = <barcodeMicroPdf417Object>:binaryMode([binaryMode])
```
binaryMode`[true|false]` sets whether a normal (false) or a binary version of the barcode should be printed.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

7.3.36 Line fields

7.3.36.1 Constructor

new()
```
line, error = lineObject.new([hPos[,vPos[,hDelta[,vDelta[,thickness[,pen]]]]]])
```
new() creates a lineObject.
If all parameters are left out the default for each parameter will be used. Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
hPos = 1
vPos = 1
hDelta = 0
vDelta = 0
thickness = 1
pen = "NORMAL"
```

7.3.36.2 Methods

pos()
```
hPos,vPos|error = <lineObject>:pos([hPos[,vPos]])
```
hPos,vPos are horizontal/vertical start position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current start position is returned.

deltaPos()
```
hDelta,vDelta|error = <lineObject>:deltaPos([hDelta[,vDelta]])
```
hDelta,vDelta are horizontal/vertical offset to end position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current end position is returned.

thickness()
```
thickness|error = <lineObject>:thickness([thickness])
```
thickness is line thickness to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current thickness is returned.

pen()
```
pen|error = <lineObject>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <lineObject>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <lineObject>:clone()
```
clone() creates an exact copy of the original lineObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <lineObject>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.37 Box fields

7.3.37.1 Constructor

new()
```
box, error = boxObject.new([hPos[,vPos[,width[,height[,thickness[,anchor
[,pen[,radius]]]]]]]])
```
new() creates a boxObject.
If all parameters are left out the default for each parameter will be used. Specific parameter(s) can be left out by writing nil at the parameter position.

error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
hPos = 1
vPos = 1
width = 1
height = 1
thickness = 1
anchor = "TOP_LEFT"
pen = "NORMAL"
radius = 0
```

7.3.37.2 Methods

pos()
```
hPos,vPos|error = <boxObject>:pos([hPos[,vPos]])
```
hPos,vPos are horizontal/vertical start position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current start position is returned.

width()
```
width|error = <boxObject>:width([width])
```
width is box width to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current box width is returned.

height()
```
height|error = <boxObject>:height([height])
```
height is box height to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current box height is returned.

thickness()
```
thickness|error = <boxObject>:thickness([thickness])
```
thickness is box thickness to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current box thickness is returned.

anchor()
```
anchor|error = <boxObject>:anchor([anchor])
```
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

pen()
```
pen|error = <boxObject>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

radius()
```
radius|error = <boxObject>:radius([radius])
```
radius sets the radius of rounded corners on the box. error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current radius is returned.
Not supported on TH2.

fieldSize()
```
hPos, vPos, width, height, error = <boxObject>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <boxObject>:clone()
```
clone() creates an exact copy of the original boxObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <boxObject>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.38 Image fields

7.3.38.1 Constructor

new()
```
image, error =
imageObject.new(path[,hPos[,vPos[,hMag[,vMag[,anchor[,pen[,dir]]]]]]])
```
new() creates an imageObject.
creates a black and white bitmap image field object from bitmap file (Microsoft BMP format), png file or pcx file pointed out by path. See Files API for definition of `<path>`, 7.1. If all optional parameters are left out the default for each parameter will be used. Specific parameter(s) can be left out by writing nil at the parameter position. Note that parameter path is mandatory.
error is set to errno.ESUCCESS if OK, errno.ENOTSUP if image is not supported, errno.EINVAL if the image file is too short or corrupt, errno.ENOENT/errno.EACCES if the file cannot be opened and otherwise errno.EPARAM. errno.ESUCCESS can be returned even for a corrupt image if the corruptedness can not be detected until rendering it.

Default values:
```
hPos = 1
vPos = 1
hMag = 1
vMag = 1
anchor = "TOP_LEFT"
```

```
pen = "NORMAL"
dir = 0
```

Supported color depths:
BMP: 1 bit/color
PCX: 1, 8 and 24 bit/color (automatic grayscale conversion).
PNG: 1, 8, 24 and 32 bit/color (Truecolor and indexed, automatic grayscale conversion, alpha channel stripped).

7.3.38.2 Methods

path()
```
path|error = <imageObject>:path([path])
```
path is the path to the image. error is set to errno.ESUCCESS if OK. errno.EPARAM is returned for invalid arguments (type or number of arguments). Other error codes can be returned as specified in new(). If called without parameter the current path is returned.

pos()
```
hPos,vPos|error = <imageObject>:pos([hPos[,vPos]])
```
hPos,vPos are horizontal/vertical position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current position is returned.

anchor()
```
anchor|error = <imageObject>:anchor([anchor])
```
anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

mag()
```
hMag, vMag|error = <imageObject>:mag([hMag[,vMag]])
```
hMag,vMag are horizontal/vertical pixel magnification to set [1..12], error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hMag and vMag are returned.

pen()
```
pen|error = <imageObject>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

dir()
```
dir|error = <imageObject>:dir([dir])
```
dir is the image rotation `[0|90|180|270]` to be used and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current direction is returned.

fieldSize()
```
hPos, vPos, width, height, error = <imageObject>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK. Otherwise the error code is set as in new().

clone()
```
clone, error = <imageObject>:clone()
```
clone() creates an exact copy of the original imageObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <imageObject>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.39 Circle fields

7.3.39.1 Constructor

new()
```
circle, error =
circleObject.new([hPos[,vPos[,diameter[,thickness[,anchor[,pen[,startAngle
                 [,stopAngle]]]]]]])
```
new() creates a circleObject.
If all parameters are left out the default for each parameter will be used. Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
hPos = 1
vPos = 1
diameter = 1
thickness = 1
anchor = "TOP_LEFT"
pen = "NORMAL"
startAngle = 0
stopAngle = 0
```

7.3.39.2 Methods

pos()
```
hPos,vPos|error = <circleObject>:pos([hPos[,vPos]])
```
hPos,vPos are horizontal/vertical start position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current start position is returned.

diameter()

```
diameter|error = <circleObject>:diameter([diameter])
```

diameter is the circle's outer diameter to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current outer diameter is returned.
The diameter of a printed circle is always an uneven number of dots. If diameter is set to an even number (2*X), then the actual diameter of the printed circle will be one less than the set value (2*X-1).

thickness()

```
thickness|error = <circleObject>:thickness([thickness])
```

thickness is circle thickness to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current circle thickness is returned. The circle grows inwards; the outer diameter of the circle is the same regardless of thickness.

anchor()

```
anchor|error = <circleObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

pen()

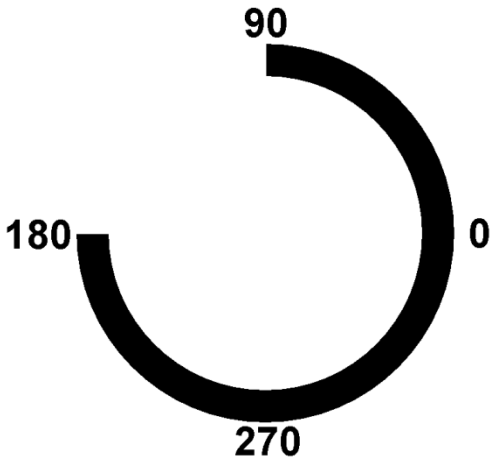
```
pen|error = <cirlceObject>:pen([pen])
```

pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

angles()

```
startAngle,stopAngle|error = <circleObject>:angles([startAngle[,stopAngle]])
```

By setting startAngle and stopAngle, it is possible to print one to four quadrants of an circle. Note that the angles can only be set to 0, 90, 180, or 270 degrees. If startAngle and stopAngle are identical, the whole circle is printed. The circle is plotted counter clockwise from startAngle to stopAngle. In the figure below, startAngle is 180 and stopAngle is 90.



error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current angles are returned.

fieldSize()
`hPos, vPos, width, height, error = <circleObject>:fieldSize()`
Returns the bounding box of the circle. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
`clone, error = <circleObject>:clone()`
clone() creates an exact copy of the original circleObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
`tbl = <circleObject>:ranges()`
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.40 Ellipse fields

7.3.40.1 Constructor

new()
```
ellipse, error =
ellipseObject.new([hPos[,vPos[,width[,height[,thickness[,anchor[,pen[,startAngle
       [,stopAngle]]]]]]]])
```
new() creates a ellipseObject.
If all parameters are left out the default for each parameter will be used. Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
hPos = 1
vPos = 1
width = 1
height = 1
thickness = 1
anchor = "TOP_LEFT"
pen = "NORMAL"
startAngle = 0
stopAngle = 0
```

7.3.40.2 Methods

pos()

```
hPos,vPos|error = <ellipseObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical start position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current start position is returned.

width()

```
width|error = <ellipseObject>:width([width])
```

width is ellipse width to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current ellipse width is returned.
The width of a printed ellipse is always an uneven number of dots. If width is set to an even number (2*X), then the actual width of the printed ellipse will be one less than the set value (2*X-1).

height()

```
height|error = <ellipseObject>:height([height])
```

height is ellipse height to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current ellipse height is returned.
The height of a printed ellipse is always an uneven number of dots. If height is set to an even number (2*X), then the actual height of the printed ellipse will be one less than the set value (2*X-1).

thickness()

```
thickness|error = <ellipseObject>:thickness([thickness])
```

thickness is ellipse thickness to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current ellipse thickness is returned.

anchor()

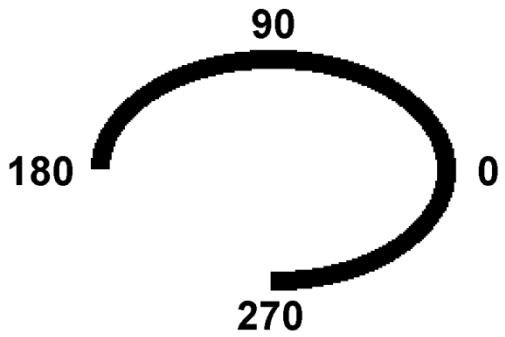
```
anchor|error = <ellipseObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

pen()

```
pen|error = <ellipseObject>:pen([pen])
```
pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

angles()
```
startAngle,stopAngle|error = <ellipseObject>:angles([startAngle[,stopAngle]])
```
By setting startAngle and stopAngle, it is possible to print one to four quadrants of an ellipse. Note that the angles can only be set to 0, 90, 180, or 270 degrees. If startAngle and stopAngle are identical, the whole ellipse is printed. The ellipse is plotted counter clockwise from startAngle to stopAngle. In the figure below, startAngle is 270 and stopAngle is 180.



error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current angles are returned.

fieldSize()
```
hPos, vPos, width, height, error = <ellipseObject>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <ellipseObject>:clone()
```
clone() creates an exact copy of the original ellipseObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <ellipseObject>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.41 Grid fields

7.3.41.1 Constructor

new()

```
grid, error =
gridObject.new([hPos[,vPos[,width[,height[,rowHeights[,colWidths[,frameThickness
               [,lineThickness[,skipLines[,anchor[,pen]]]]]]]]]])
```

new() creates a gridObject.
If all parameters are left out the default for each parameter will be used. Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:

```
hPos = 1
vPos = 1
width = 100
height = 100
rowHeights = {}
colWidths = {}
skipLines = {"horiz":[],"vert":[]}
frameThickness = 1
lineThickness = 1
anchor = "TOP_LEFT"
pen = "NORMAL"
```

7.3.41.2 Methods

pos()

```
hPos,vPos|error = <gridObject>:pos([hPos[,vPos]])
```

hPos,vPos are horizontal/vertical start position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current start position is returned.

width()

```
width|error = <gridObject>:width([width])
```

width is the grid width to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current grid width is returned.

height()

```
height|error = <gridObject>:height([height])
```

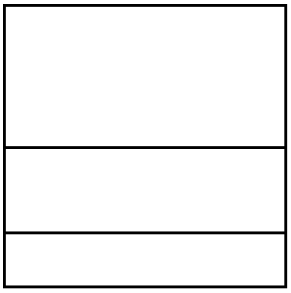
height is grid height to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current grid height is returned.

rowHeights()

```
rowHeights|error = <gridObject>:rowHeights([rowHeights])
```

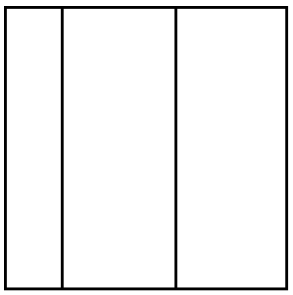
rowHeights is a Lua table that defines the heights of the rows in the grid. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current tabel is returned.

Example:
```
g=gridObject.new()
g:rowHeights({50, 30})
```
This will create two rows. The one at the top is 50 dots high, the next is 30 dots high (minus frame and line thickness).
Rows that do not fit within the total height of the grid will not be drawn. Remaining space, if any, below the last defined row will become the bottom row.



colWidths()
```
colWidths|error = <gridObject>:colWidths([colWidths])
```
colWidths is a Lua table that defines the widths of the columns in the grid. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current tabel is returned.

Example:
```
g=gridObject.new()
g:colWidths({20, 40})
```
This will create two columns. The leftmost is 20 dots wide, the next is 40 dots wide (minus frame and line thickness).
Columns that do not fit within the total width of the grid will not be drawn. Remaining space, if any, to the right of the the last defined column will become the rightmost column.



frameThickness()
```
frameThickness|error = <gridObject>:frameThickness([frameThickness])
```
frameThickness is frame thickness to set (the border lines), error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current frame thickness is returned.

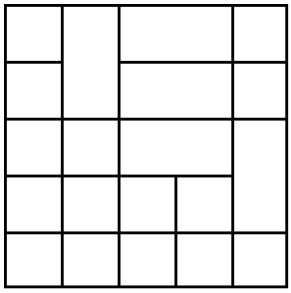
lineThickness()
```
lineThickness|error = <gridObject>:lineThickness([lineThickness])
```
lineThickness is line thickness (horizontal and vertical) to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current line thickness is returned.

skipLines()
```
skipLines|error = <gridObject>:skipLines([skipLines])
```
skipLines is a Lua table that sets which vertical and horizontal lines in the grid not to draw. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current skipLine table will be returned. If called with 'nil', the previous table of lines to skip will be deleted.

Example:

```
g=gridObject.new()
g:colWidths({20,20,20,20})
g:rowHeights({20,20,20,20})
g:skipLines({["horiz"]={["1"]={2},
["3"]={5}},["vert"]={["3"]={1,2,3}}})
```

This means that horizontal line 1 part 2, horizontal line 3 part 5, and vertical line 3 part 1-3 will not be drawn.



mergeCells()

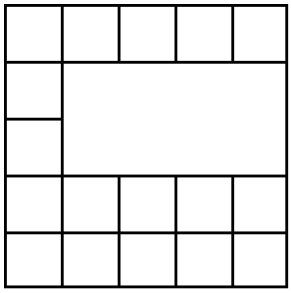
```
error = <gridObject>:mergeCells({row1, column1, row2, column2})
```

mergeCells is a Lua table that defines grid cells to merge.
This command will affect the skipLines table.

Example:

```
g=gridObject.new()
g:colWidths({20,20,20,20})
g:rowHeights({20,20,20,20})
g:mergeCells({2,2,3,5})
```

This means that the cells from row 2, column 2 to row 3, column 5 are merged.



splitCells()

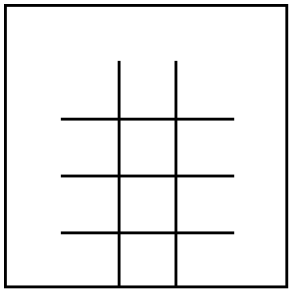
```
error = <gridObject>:splitCells({row1, column1, row2, column2})
```

splitCells is a Lua table that defines grid cells to be split.
This command will affect the skipLines table.

Example:

```
g=gridObject.new()
g:colWidths({20,20,20,20})
g:rowHeights({20,20,20,20})
g:mergeCells({1,1,5,5}) -- Merge all
g:splitCells({2,2,5,4})
```

This means that the cells from row 2, column 2 to row 5, column 4 are split again. Note that all cells were merged before the split.



anchor()

```
anchor|error = <gridObject>:anchor([anchor])
```

anchor is anchor point to be used `["TOP_LEFT"|"TOP_CENTER"|"TOP_RIGHT"|"MID_LEFT"|"MID_CENTER"|"MID_RIGHT"|"BASE_LEFT"|"BASE_CENTER"|"BASE_RIGHT"|"BOTTOM_LEFT"|"BOTTOM_CENTER"|"BOTTOM_RIGHT"]` and error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current anchor is returned.

pen()

```
pen|error = <gridObject>:pen([pen])
```

pen is the pen mode, ["NORMAL"|"REVERSE"|"ERASE"|"REPLACE"], used when printing the object. Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current pen is returned.

fieldSize()
```
hPos, vPos, width, height, error = <gridObject>:fieldSize()
```
Returns the bounding box of the object. `hPos,vPos` is the upper-left corner and width, height the width and height of the bounding box. `error` equals errno.ESUCCESS on OK, else errno.EPARAM.

clone()
```
clone, error = <gridObject>:clone()
```
clone() creates an exact copy of the original gridObject.
Error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.ENOMEM.

ranges()
```
tbl = <gridObject>:ranges()
```
Not available on TH2.
ranges() returns a table with all attributes associated with the object and all the attributes ranges including any options the attribute might have. For example, see 7.3.2

7.3.42 Label

7.3.42.1 Constructor

new()
```
label, error = labelObject.new([hBase[,vBase]])
```
new() creates a labelObject.
If all parameters are left out the default for each parameter will be used. Specific parameter(s) can be left out by writing nil at the parameter position.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default values:
```
hBase = 0
vBase = 0
```

7.3.42.2 Methods

add()
```
fieldNo, error = <labelObject>:add(object[,err])
```
object is any of the above mentioned objects, error is set to errno.ESUCCESS if OK and fieldNo is assigned an unique object field number within the label, otherwise error is errno.EPARAM. If object is nil, err parameter will be propagated out to error. This feature can be used in a special case were an object constructor is used inside the add functions, see 7.3.43.
If called without parameters errno.EPARAM is returned.

remove()

```
error = <labelObject>:remove(fieldNo)
```
fieldNo is the unique object assigned field number of the object to be removed from the label object, error is set to errno.ESUCCESS if OK, errno.ENOTFOUND if field number not found and errno.EPARAM if no parameter.

render()
```
<labelObject>:render(<canvasObject>)
```
Draw label in printer image buffer derived from <canvasObject>.

base()
```
hBase,vBase|error = <labelObject>:pos([hBase[,vBase]])
```
hBase,vBase are horizontal/vertical offset position to set, error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current hBase and vBase are returned.

7.3.43 Example

```
-- Create and print a label

-- create an upside down text field
text = textTTObject.new(nil,"Hello World!",200,100)
text:dir(180)
-- create and add a barcode Code 39 field with height 25
bc = barcodeObject.newCode39("123456")
bc:height(25)
-- create label and add the above fields
label = labelObject.new()
label:add(text)
label:add(bc)
-- add an EAN-8 field
label:add(barcodeObject.newEan8("1234567",50,250))
-- render and print
canvas = engine.newCanvas()
canvas:render(label)
canvas:print()
```

7.3.44 Possible future enhancements

7.3.44.1 Rotation of text

Some users may want to rotate text elements.
If direction (dir) is set to 270 and rotation is set to 90, the text is printed like this.
This will not be implemented.

L
u
a

7.3.44.2 Multiple properties for barcode width

The SBPL commands BT and BW make it possible to set the width of narrow space, narrow bar, wide space and wide bar for a few barcodes.

This is considered to be unnecessary and will not be implemented.

7.3.44.3 Positioning unit

The unit of all position parameters is dots.

It may be a good idea to let the user select which positioning unit to use.

If the unit could be set to mm, an application could run on two printers with different print head resolution and still get the same printed result.

7.3.44.4 Line pattern

A pattern parameter could be introduced to change the appearance of line and box objects. Pattern could either be selected from a number of defined choices (e.g. "solid", "dotted", "dashed",...) or simply an integer defining a bitmap.

7.4 Engine functions

In the engine API, functions to do feeding and preparing printing is found. For a higher level interface, see 7.5 - "Print Job Handler".

7.4.1 Functions

feed()

```
jobStatusObject[, error] = engine.feed(<dots> [,<queue-style>])
```

This function feeds paper as many dots as the given argument. The function can return two error types: errno.EPARAM if <dots> is 0 and errno.ENOMEM if the statusObject could not be created. The feed is executed in the background. The feed does not start until the printer is online (NB! TH2 is always online).

formFeed()

```
jobStatusObject[, error] = engine.formFeed([<queue-style>,<qty>,<cut>])
```

This function feeds one form.
The function takes optional qty-argument to feed the specified number of forms.
The optional cut-argument specifies after how many forms a cut should be issued. The default parameters are: engine.DO_FIFO, 1,0.
When the queue-style is engine.DO_FIFO, the feed does not start until the printer is online .

sensorDrive()

```
<oldImDrive>, <oldGapDrive> = engine.sensorDrive(<imDrive>,<gapDrive>)
```

This function is used to set and/or read the current drive for I-Mark and Gap diode. If called with no arguments current drives are returned. If called with illegal or too few or to many arguments nil, nil is returned. This is for TH2 only.
Valid drives are 0<=imDrive<=3, 0<=gapDrive<=3.

sensorLevels()
sl = engine.sensorLevels(<bClear>)
Returns a table with samples taken, minimum and maximum A/D values for gap and I-Mark sensor since last clear. Values for not selected sensor type is shall be ignored. This is for TH2 only.
samples samples taken

imMin min detection by the I-mark sensor
imMax max detection by the I-mark sensor
gapMin min detection by the GAP sensor.
gapMax max detection by the GAP sensor (median filter used. Max will represent label level for the GAP sensor).

bClear false to clear and true to return the collected minimum and maximum values. When called with illegal argument nil is returned.

sensorCalLevels()
imMin, imMax, gapMin, gapMax =
engine.sensorCalLevels(<newImMin>,<newImMax>,<newGapMin>,<newGapMax>)
Function to read/set the calibration settings. Settings are not stored in between power cycles. This is for TH2 only.

canvasInfo()

```
t = engine.canvasInfo()
```

This function returns a table with count, free, length and maxLength that gives the total number of canvases, the free number of canvases, the length of each canvas (in dots) and the maximum allowed length of each canvas (in dots). The count-parameter is for the application to manage resources. The others are mostly for debug-purposes.

newCanvas()

```
-- a printable canvas
canvasObject, error = engine.newCanvas()

-- non-printable canvas
canvasObject, error = engine.newCanvas(true)
canvasObject, error = engine.newCanvas(<canvasObject>)
canvasObject, error = engine.newCanvas(dpmm,width,height)
```

This function returns a canvasObject to draw the bitmap into on success and nil on failure. Failure is because the application holds old canvases or old statusObjects. The canvasObject returns a cleared canvas. The TH2 supports only printable canvases. The non-printable canvases are useful for getting drawing areas for other purposes. Three resolutions for dpmm are supported: 8,12,24; width must be a multiple of 32 and height must be at least 32. If true is used, the printers resolution,width and current drawing length is used.

toTear()

```
statusObject[, error] = engine.toTear([<queue-style>])
```

This function is used to feed paper for tear off purposes. It works differently depending on the current media handling. If the media handling is “tear off” and the engine is waiting, it will stop waiting and feed to the tear bar. If the media handling is “continuous” and the last job was “print” or “formfeed” it will feed to the tear bar. At the start of the next job, the motors will backfeed to dot row again before printing. Otherwise (including other media modes), it will behave as a engine.formFeed(). This is only supported in TH2.

cut()

```
statusObject[, error] = engine.cut([<queue-style>])
```

This function is used to run a cut cycle. This is useful in the TH2 if the engine has a cutter but has not enabled cutter media handling. The cut function is also useful in cases where cut is suppressed until a batch is completed. NB! The cut operation is only supported after printing/feeding a label where the cut would normally occur unless it was suppressed.

pause()

```
engine.pause()
```

If multiple jobs are queued up in the engine, this function provides a method to shutdown the jobs after the current job is finished. This function is only in TH2.

count()

```
<table>, <error> = engine.count()
```

Returns a table with counters in TH2 only. The attributes are as follows:

`life` - total number of meters the printer has fed and/or printed.

`head` - number of meters the current head has fed and/or printed.

`head1` - number of meters the previous head has fed and/or printed.

`head2` - number of meters the 2nd previous head has fed and/or printed.

`cut` - number of cuts the current cutter has started.

resume()

```
engine.resume([bReinit])
```

TH2:

This method clears the shutdown status and is used to acknowledge the error before resuming or cancelling jobs if called without arguments.

If it is called with a true value of type bool, it reinitializes the sensors. It is used to make the print engine forget all saved samples to guarantee that errors introduced in label positioning between power cycles are eliminated.

Other printer models:

This method is internally used to clear the cancel job flag.

skipMode()

```
mode,error = engine.skipMode([enable])
```

This method controls the behavior when the printer cannot fit the printout on the label. The default behavior is to print the remaining dot-lines and then feed out the label. When the printout barely fits the label, the effect is that the printer skips labels (sometimes feeds an extra label).

The skip behavior can be reduced by setting it to false. The printer will then favor excluding to print the remaining dot-lines (as many as fits in 5mm). If more than that remains to be printed more labels will be used until satisfied.

The default skipMode is enabled for backward compatibility. This is supported only in TH2.

cancelHandshake()

```
status[, error] = engine.cancelHandshake()
```

This is an internal function used to fulfill the cancel job protocol. It returns errno.ESUCCESS on success and nil, error on failure. This is not supported in TH2.

headInfo()

```
info = engine.headInfo()
status = engine.headInfo{dpmm=dpmm,width=width,anchor=anchor}
```

This is mostly an internal function to emulate print heads. The anchor parameter can be center |right and describes the physical anchoring for the mounted label roll. This is not supported in TH2.

mainshare(),mctl([i]), shrvar()

```
mainshare = engine.mainshare()
mctl = engine.mctl([i])
shrvar = engine.shrvar()
```

Internal functions to read out the members of shared memory, for debugging. This is not supported in TH2.

mstat()

```
mstat = engine.mstat([mstat])
```

Internal function to write/read the members of shared memory, for debugging. This is not supported in TH2.

qty()

```
status,error = engine.qty(qty[,id[,wk]])
```

This is function is used to set the Quantity information for the LCD. This is not supported in TH2.

queueFull()

```
status,cancelStatus = engine.queueFull()
```

This internal function can be used to read the queue status and cancel flag. This is not supported in TH2.

reclaim(),restore()

```
failedReclaims = engine.reclaim()
inUse = engine.restore()
```

These functions are required for cooperation with the SATO emulators and DC2. Before handing over control to the emulator, call restore(). After resuming AEP operations, call reclaim(). The effects of the emulator may be that it reuses a canvas AEP allocated, and such a canvas is considered dirty and is blocked from printing. The return values from reclaim(),restore() indicate how many canvases that AEP failed to reclaim and the number of canvases AEP had in use at restore().This is not supported in TH2.

aepwDesign()

```
design_width,design_height= engine.aepwDesign([false]|[width,height])
```

This function is used to set the drawing area for AEP formats non-intrusive to the rest of the system. This is not supported in TH2.

prin()

```
idx= engine.prin()
```

This function is used internally to trigger "realtime print" and "wait for EXTIO Start Print". This is not supported in TH2.

7.4.2 Engine constants

The engine constants define the queue-style parameter.

`engine.DO_FIFO`
This queue-style is the default queue-style and it makes the engine execute jobs in First-In-First-Out order. If a job is received when the engine is shutdown, it will not start until it is being resumed.

`engine.DO_ASAP`
This queue-style makes the engine execute the job As-Soon-As-Possible. If the engine has been shutdown, and the current status is OK, it will execute the job as soon as the executing job has fininshed. If the engine is shutdown before it starts it needs to be resumed.

7.4.3 Engine shutdown

TH2: If the print engine detects an error it is shutdown. The shutdown status remains until it is explicitly cleared by the resume method or by registering a job that ignores the shutdown status.

7.4.4 Canvas methods

clear()
```
status = <canvasObject>:clear()
```
This method clears the canvas if it is not busy. The canvas is busy if it is being printed or if a statusObject contains a reference to it.

render()
```
<canvasObject>:render(<labelObject>[, …])
```
This method takes 1..N labelObjects and renders them into the canvas's image buffer.

print()
```
jobStatusObject, error = <canvasObject>:print([<queue-style>])
jobStatusObject, error = <canvasObject>:print([<queue-style>][,copies[,cut]])
jobStatusObject, error = <canvasObject>:print([<queue-
style>][,copies[,cut][,idx]])
```
This method returns a statusObject on success and starts printing the canvas in the background. The print starts first when the printer is online (NB! TH2 is always online).
The copies and cut arguments specifies the number of copies and how many to count before cutting. The default parameters are: _,1,1,nil

errno.EFAULT - error returned for a non-printable canvas.
errno.ESTALEID - error returned for a canvas that AEP failed to reclaim (see restore/reclaim).
errno.ENOTHINGTOPRINT - error for a canvas that had no rendered data.

The idx-parameter is used for realtime-print and the value from engine.prin() should be used.

save()
```
status = <canvasObject>:save(<path>)
status = <canvasObject>:save(<path>,<from>,<to>[,limit])
BMP_imagedata[,err]=canvas:toBmp([startRow[,endRow[,width]]])
```

This saves the canvas's image buffer into a file with Windows Monochrome BMP file format.
The optional arguments control from which line to which line and the boolean limit can be used to limit the width to the part that is rendered.
The toBmp() method does not write to file. It returns the image as a string instead.

suppressOffsets()
```
suppressOffsets|error = <canvasObject>:suppressOffsets([true|false)
```
If suppressOffsets is true, then the horizontal and vertical position of rendered objects are not affected by settings Imaging->Vertical/Horizontal or Label Width.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.
If called without parameter, current value of suppressOffsets is returned.
Default value is false for printable canvases and true for non-printable canvases.
Note! This method is not supported in TH2.

7.4.5 jobStatusObject methods

id()
```
id = <jobStatusObject>:id()
```
This method returns the id of the job that it represents. The id is a number that starts from 1 and counts upwards.

resume()
```
<jobStatusObject>:resume()
```
In TH2 only this method resumes a job that was done or stopped. The job can be of any type. The job id is consistent for resumed jobs, so the resume()-method should only be used to recover after errors.

status()
```
status, errorCode = <jobStatusObject>:status([flag])
```
This method returns the bitpattern-status and errorCode a job has set. It is only the job that was stopped by an error that gets an errorCode as non-zero. If flag is passed it will test specifically against that condition, or else the status must be interpreted as a bitpattern.

7.4.6 jobStatusObject constants

The jobStatusObject holds constants for testing the status of the job. These constants below are defined.

JOB_notStarted
```
<statusObject>.JOB_notStarted
```
This constant value is set at registration and indicates that the job has never started.

JOB_started
`<statusObject>.JOB_started`
This constant value is set when the job is starting to be processed.

JOB_printing
`<statusObject>.JOB_printing`
This constant value is set while the job is being printed, and then it is cleared.

JOB_printed
`<statusObject>.JOB_printed`
This constant value is set after the job has been printed.

JOB_wait
`<statusObject>.JOB_wait`
This constant value is set while the print engine waits for something. It is cleared when it is not waiting anymore. It could be waiting for a tear off timeout, a label to be removed (dispenser/peel off) or for the cutter cycle to finish.

JOB_done
`<statusObject>.JOB_done`
This constant value is set when the print engine is all done with the job. This means at the time which it transfers control to the next job or go idle.

JOB_dispense
`<statusObject>.JOB_dispense`
This constant value is set when the print engine starts to wait for label dispense.

JOB_cut
`<statusObject>.JOB_cut`
This constant value is set when the print engine starts the cut cycle.

JOB_paused
`<statusObject>.JOB_paused`
This constant value is set if the job was shutdown because the engine was paused.

JOB_offsetTooLow
`<statusObject>.JOB_offsetTooLow`
This constant value is set if the pitch offset would cause the print engine to backfeed. It will also generate EPITCHERROR.

JOB_labels
`<statusObject>.JOB_labels`
This constant value is set if the pitch offset is so low that it interferes with printing the entire image. When set, the engine will stop where the printing stops.

JOB_printOffsetOut

<statusObject>.JOB_printOffsetOut
This constant value is set if the print offset is so low that there are no lines left to print.

7.4.7 System status

status()
status, errorCode = system.status()
This function is used to get information about the overall system status. The status value is a bitpattern that can be tested with the symbolic names in Table 2. The errorCode is the error number that stopped the engine.

Table 2. Meaning of status (numeric constants)

Symbolic name	Meaning	
system.SYS_jobStatus	A jobstatusObject has been updated.	
system.SYS_isMoving	The printer is moving paper	
system.SYS_coverOpen	The cover is open	
system.SYS_noPaper	The printer is cannot detect paper	
system.SYS_hasCutter	A cutter is detected	
system.SYS_cutterIsHome	The cutter is at its home position.	
system.SYS_usbConnected	USB connected	
system.SYS_linkDown	Network link is down	Status is unknown if none of them is set.
system.SYS_linkUp	Network link is up	

7.5 Print Job Handler

The Print Job Handler, contained in the job table, provides a higher level access to the print functionality than the engine interface. The job handler provides a number of functions.

job.add()
<remain>, <aborted>, <err> = job.add(<canvas>,[<max>])
<remain>, <aborted>, <err> = job.add(<canvas>,[<max>],[cbAdd], [cbDone], [cut], [idx])
This function queues up a new print job.
The parameter <canvas> is a rendered canvas.
Optionally, a parameter <max> can be added. This is to mark the beginning of a batch of <max> print jobs. This number is used in the LCD to indicate the batch progress. If <max> is not provided at the beginning of a new batch, the printer will update the batch counter when new print jobs are added. <remain> is the number registered remaining jobs, <aborted> is true if the user has aborted the batch and <err> is set to the error number causing the batch to be stopped (errno.ESUCCESS if it was paused by the user). The cut or idx parameters are not supported in TH2.
cbAdd and cbDone are callback functions called at add and at job done. To supress cutting after each job.add, pass cut=false. If a canvas has no rendered lines, job.add will add it as a engine.formFeed()-request to feed a blank label.
In "real time print"-mode the idx-parameter is retrieved from job.waitStart().
The max parameter should be nil,number or a table containing {max=max,copies=copies} ; The table type is not supported in TH2.

job.addFeed()
`job.addFeed([<dots>])`
This function queues up a new feed job.
If the parameter `<dots>` is provided, the job will feed `<dots>` number of dots.
If no parameter is provided in TH2, the job will feed using `engine.toTear()`. Other models use engine.formFeed()

job.poll([true])
`<remain>, <aborted>, <err> = job.poll([true])`
This function checks the status of the print and feed jobs in the queue and returns the number of remaining jobs and a status parameter (true or false). If the status parameter, `<aborted>`, is true, this means that the user has cancelled all jobs in the queue. `<err>` is set to the error number causing the batch to be stopped (`errno.ESUCCESS` if it was paused by the user). In TH2 this function doesn't take any arguments, but for other printers, true can be passed to guarantee an immediate return.

job.blockLowBattError(block)
This function is used internally to turn off low battery error when printing test labels that exceed the battery load capacity, because it contains too much black. This is TH2 only.

job.stats
{total=n,clips=n,skips=n}
This is a Lua-table that is maintained by the Print Job Handler. After printing/feeding total reflects the number of jobs executed, clips the number of times a printout has been clipped and skips how many times the print engine has used an extra label because the printout didn't fit a physical label. The clips and skips are affected by the setting of engine.skipMode(). This is TH2 only.

job.addEventsAndHandler()
`subscribeTable = job.addEventsAndHandler(callbacks,handler)`
This function registers callbacks recommended to handle events firing at printing and a handler called by job to avoid busy wait operations.

job.runSbplFromAep()
`currentFunction=job.runSbplFromAep([<cbFunction>])`
This function makes it possible to run SBPL commands from AEP. AEPWorks can be used as usual to create a layout, but the string returned from function `cbFunction` will be interpreted by SBPL.

Example: To add a triangle rendered by SBPL to your layout, add these lines.

```
  function PrintTriangle()
  return "\27H200\27V100\27FT,200,30,200,0"
end
job.runSbplFromAep(PrintTriangle)
```

All labels rendered will now include the triangle defined by the function PrintTriangle.
To change the commands sent to the SBPL interpreter, just call

job.runSbplFromAep(withAnotherCbFunction) i.e. another lua function that returns the SBPL you prefer. To disable this functionality, call `job.runSbplFromAep(false)` again.

There are some limitations to printing in this mode:

For QTY n>1, it shows n times QTY 1 as AEP issues one SBPL item at a time. This blocks serial numbering in SBPL. AEP counters are not affected.

Managing "real time" printing and EXTIO Start Print is handed over to SBPL, so it's better to use SBPL calendar functions for printing time fields.

Note!
All fields rendered by AEP use AEP settings and all fields rendered by SBPL use SBPL settings. Specifically, the CLNX setting:
- Application -> AEP -> Label rotation: affects AEP fields
- Application -> SBPL -> Orientation: affects SBPL fields
- Printing -> Label Length: affects SBPL fields
- Printing -> Label Width: affects SBPL fields

AEP fields get media length and width from settings in the format.

This function is not supported in TH2.

job.waitstart()

```
idx = job.waitStart()
```

This function is used for "realtime print"-mode and for EXTIO. This is not supported in TH2.

job.waits ()

```
true|false = job.waits()
```

This function returns true when job.waitStart() waits for a "start signal". This is not supported in TH2.

job.pstate

```
job.pstate
```

The job.pstate-table contains runtime configuration variables for job. Some of them can be set to override standard behavior:

```
job.pstate.LTS -- default true implies to wait in dispensing mode
job.pstate.waits -- default true affects job.waits() with EXTIO-enabled
job.pstate.realtime -- default false. Set to true to enable job.waits() without
EXTIO.
```

To change the default behavior, change from true to false or vice versa. This is not supported in TH2.

7.5.1 Callbacks in TH2

The job handler calls a number of functions that can be overridden, should the application want to do something else than the default. The default behavior is also described. NB! This is TH2 only.

job.setup()
Called when the user presses F2 to enter the printer setup.
Default behavior is to make sure config.setup() is loaded and call it (inhibiting printouts, as they might cause another error).

job.errorHandling(err, count, max)
Called if there is an error printing a canvas or if the user pauses the batch.
`err` is the error number (errno.ESUCCESS on user pause), `count` the number of the job causing/affected by the error and `max` is the total number of jobs in the batch.
Default behavior is basically to display the error and provide a help screen, making it possible for the user to enter the setup, cancel the batch, feed labels, resume or cancel the entire batch. See ref [4].

job.batchCounter(count, max, start)
Called multiple times during batch printing, its function is to write the batch counter to the display. At start of batch (`start == true`) the first row in the LCD is saved. At end of batch (`start == false`) the first row is restored. During a batch it is called with `start == nil`. `count` is the current job to be printed and `max` is the total number of jobs in the batch.

job.dispenseAnim([<j>])
This function is used to animate the peel-off images at the application icon.

job.numOfPrintJobs()
This function is used to determine how many of the queued jobs that are print jobs.

7.5.2 Usage

This is a typical way to use the error handling functions described in pseudo code.

```
loop
          input data etc
          create canvas and render
          job.add(canvas)                 -- Add print job to queue
          job.poll()                      -- Not necessary if loop is fast
until ready

do other stuff

repeat
          do other stuff
until job.poll() == 0           -- Wait here until all jobs are done
```

7.6 Configuration

The printer configuration is accessible through the config library and the Lua table configTbl. The config library contains functions for reading and storing configurations in Lua and XML format. Reading/Writing configuration values are done by reading/writing to the Lua table called configTbl,

see [0]. XML format follows the XML schema located at /rom/schemas/configTblSchema.xsd in the printer's file system.

7.6.1 Functions

write()
`error = config.write([<path|Lua File>][, group][, include][,filter])`
`<path>` is the complete file path for the Lua config data file or a Lua File object (e.g. io.stdout). error is set to errno.ESUCCESS if ok, else errno.EPARAM or errno.ENOENT if failed to create file. If called without path, config data file is stored at default location, /ffs/config/config.lua. If called with a path, file system access applies to destination folder.

`<group>` A bit pattern or `nil` for default that specifies what groups to include/exclude.

`<include>` If set to `true` or `nil`, only the specified groups are included in the output file. If set to false, the groups are excluded. Default is `true`.

`<filter(token)>` A Lua function that receives a string `token` (e.g. `"pdd.speed"` for `configTbl.pdd.speed`) for each setting about to be written. Return `true` to include it and `false` to exclude it.

read()
`error = config.read([<path>])`
`<path>` is the complete file path for the Lua config data file. error is set to errno.ESUCCESS if loaded and configured correctly, else errno.EPARAM or NOENT if failed to open file. If called without path, config data file is read from default location, /ffs/config/config.lua.

writeXML()
`error = config.writeXML([<path>])`
`<path>` is the complete file path for the XML config data file. error is set to errno.ESUCCESS if ok, else errno.EPARAM or errno.ENOENT if failed to create file. If called without path, config data file is stored at default location, /ffs/config/config.xml. If called with a path, file system access applies to destination folder. This function is only supported in TH2.

readXML()
`config.readXML([<path>])`
`<path>` is the complete file path for the XML config data file. If called without path, config data file is read from default location, /ffs/config/config.xml. This function is supported only in TH2.

config.createProfile()
error = config.createProfile([<filename>])
`<filename>` is the name of the profile and should have the extension '.lua'. error is set to errno.ESUCCESS if ok, else errno.EPARAM or errno.ENOENT if failed to create file. Function restricted to admin and manager. Profiles are created in /ffs/config/profiles/. This function is supported only in TH2.

config.removeProfile()
error = config.removeProfile([<filename>])
<filename> is the name of the profile and should have the extension '.lua'. error is set to errno.ESUCCESS if ok, else errno.EPARAM or errno.ENOENT if failed to remove file.

Function is restricted to admin or manager, unless called with the default config path (/ffs/config/config.lua) which will remove the current configuration regardless of current user. Profiles are created in /ffs/config/profiles/. This function is supported only in TH2.

config.xmlToLua()
status, errno = config.xmlToLua(<xmlPath>, <LuaPath>)
Converts an XML setup file, <xmlPath>, to a Lua file, <LuaPath>. status is true on success, and nil otherwise. error is set to errno.ESUCCESS if ok, else the error number. This function is supported only in TH2.

config.reset()
status, errno = config.reset([<groups to exclude from reset>][, <exclude>])
Reset settings. This function is not supported in TH2. When no arguments are given, a user reset is performed. Valid groups are defined in the table config.grp. Valid arguments are shown in the table below:

Argument 1	Argument 2	Description
config.grp.RST_N	<none>	Performs User Reset. Translates to config.grp.RST_F+config.grp.RST_KF+config.grp.RST_K +config.grp.RST_KU
config.grp.RST_F	<none>	Performs Factory Reset. Translates to config.grp.RST_F+config.grp.RST_KF+config.grp.RST_KU
config.grp.RST_GI	false	Resets Interface group only
config.grp.RST_GP	false	Resets Printing group only

To exclude multiple groups, the groups can be added.
NB! This function can only be executed by root.

config.getStructure()
t = config.getStructure()
This internal function returns the configuration structure as a table. This function is not supported in TH2.

config.getByPath(), config.rmByPath(), config.getById(), config.rmById(),config.byEach()
t,pos = config.getByPath(table,path)
t = config.rmByPath(table,path)
t,pos = config.getById(table,id)
t = config.rmById(table,id)
n=config.each(table,props,function)
These internal functions are used to modify the configuration tree. They operate on a table containing the configuration structure and then locates the requested item from path or id. If the item is found, it is returned in t; otherwise t is nil. The table position is returned in pos. NB! The path is translated to the current language specified. These functions are not supported in TH2.

config.decorate()
config.decorate(t,decore,recursive,replace)
This internal function applies the properties in the table decore to the table t in leaves having attribute score or value. The properties are appended unless replace is true. The table t can be recursively traversed to decorate deeper nodes when recursive is true. This function is not supported in TH2.

config.xlate()
l,enum=config.xlate(k,eigo)
This function performs a translation of key k to the current language setting unless eigo is true. Eigo is an override to translate it to US English. The translation is in l and enum is the symbolic name for error codes. This function is not supported in TH2.

config.getLocalizedUnit()
str,unit,div=config.getLocalizedUnit(t)
This internal function will convert dots to inches or mm depending on the current locale setting and return as a string with up to 3 decimals. This function is not supported in TH2.

config.req()
last_req=config.req([set,req])
This internal function returns the req parameter, which contains information about the type of request done. This is used do differentiate between web and local requests. This function is not supported in TH2.

config.proxy()
r,err=config.proxy(code,fn,arg,timeout)
This internal function is used to execute the function fn in code code with the arguments arg. A response is waited for maximum timeout seconds. This function is not supported in TH2.

config.getOts()
t=config.getOts(path,lang)
This function is used by SOS to get the settings structure. The settings are localized according to lang. The lang-parameter is "ja" for Japanese, "sv" for Swedish and "en_US" for English, US. Any supported language is possible. This function is not supported in TH2.

config.getVideos()
t=config.getVideos()
This function is used by SOS to get a table with pairs of urls+help video names. This function is not supported in TH2.

config.setOts()
stats=config.setOts(t)

This function is used by SOS to set settings in t, which is either a json-string or a lua table. The stats is a table containing the counts for total,set and error which depends on the data in t. This function is not supported in TH2.

config.screenshot()
pngData=config.screenshot()
This function is used to capture a screenshot of the LCD and return it as a PNG-image. Used in WebConfig. This uses screenshot.lua which will be used by SOS. This function is not supported in TH2.

config.clearCache()
config.clearCache()
This internal function is used to reset configRange and reload ranges from C. This function is not supported in TH2.

7.6.2 Table

The configTbl is available in the global Lua scope at boot. Writes will update the f/w and the table if value is valid. The configTbl doesn't accept additions of keys (nodes). This is an example from TH2, and matches only partially with other models.

```
configTbl.profile.select = ""
configTbl.startApp = "/rom/standalone/sa.lua"
configTbl.network.wlan.ssid = "SATO"
configTbl.network.wlan.adhocSecType = "none"
configTbl.network.wlan.infraSecType = "none"
configTbl.network.wlan.wpaType = "eap"
configTbl.network.wlan.eap.passw = "****"
configTbl.network.wlan.eap.mode = "PEAP"
configTbl.network.wlan.eap.uname = ""
configTbl.network.wlan.wep.index = 1
configTbl.network.wlan.wep.key2 = ""
configTbl.network.wlan.wep.key1 = ""
configTbl.network.wlan.wep.key3 = ""
configTbl.network.wlan.wep.key4 = ""
configTbl.network.wlan.wep.auth = 0
configTbl.network.wlan.channel = 1
configTbl.network.wlan.mode = "adhoc"
configTbl.network.wlan.wpa2Type = "eap"
configTbl.network.wlan.wpa.psk = "****"
configTbl.network.active = false
configTbl.network.lan.gateway = "000.000.000.000"
configTbl.network.lan.mode = "DHCP"
configTbl.network.lan.netmask = "000.000.000.000"
configTbl.network.lan.ip = "000.000.000.000"
configTbl.media.maxFeed = 1872
configTbl.media.sensorType = "I-MARK"
configTbl.media.size.width = 448
configTbl.media.size.length = 608
```

```
configTbl.bluetooth.discoverable = true
configTbl.bluetooth.pincode = "0000"
configTbl.bluetooth.name = "SATO Printer"
configTbl.bluetooth.security = 0
configTbl.regional.dst = true
configTbl.regional.zone = "1"
configTbl.regional.language.locale = "/rom/locales/en.all/"
configTbl.regional.language.messages = "/rom/locales/en.all/"
configTbl.regional.language.keyboard = "/rom/locales/Full/"
configTbl.regional.language.ps2 = "/rom/locales/en.all/"
configTbl.regional.unit = "dot"
configTbl.printControl.autoFeed.afterError = false
configTbl.printControl.autoFeed.powerOn = false
configTbl.printControl.speed = 4
configTbl.printControl.image.rotation = 0
configTbl.printControl.image.mirror = false
configTbl.printControl.image.offset.horizontal = 0
configTbl.printControl.image.offset.vertical = 0
configTbl.printControl.tearOffDelay = 0.500
configTbl.printControl.darkness = 3
configTbl.printControl.adjustment.pitch = 0
configTbl.printControl.adjustment.offset = 0
configTbl.printControl.adjustment.posAdjust = 0
configTbl.printControl.backfeedMode = "BEFORE"
configTbl.printControl.mediaHandling = "TEAR OFF"
configTbl.printControl.headCheck = "ALL"
configTbl.system.display.intensity = 24
configTbl.system.display.backlight = true
configTbl.system.sound.error = true
configTbl.system.sound.keyboard = true
configTbl.system.autoOff.AC = 0
configTbl.system.autoOff.battery = 0
```

Ex:

```
configTbl.media.sensorType = "GAP" -- use GAP sensor
```

7.6.3 Navigation

Configuration navigation (setup) is available by calling the Lua function config.setup(). All display text is wrapped through the current translation table, see [8.1]. The setup functionality is not available at boot and must be loaded by user (/rom/setup/setup.lua"). This sections describes TH2.

Navigation keys:

Enter enter node or validate input. If at leaf and no change leave node.

PgUp leave node without validate and set data. Pressed more then 2 sec will leave navigation.

Down Arrow move down in selection list and if at last to first. If pressed more than 1 sec jump to last.

Up Arrow move up in selection list and if at top to last. If pressed more than 1 sec jump to first.

Left Arrow step left one character or if pressed 1 sec jump to first.

Right Arrow step right one character or if pressed 1 sec jump to last.

7.6.3.1 Functions

setup()

config.setup([extensionTbl][,hide][,downloadCheckOverride])

This function is supported only in TH2. If called without the extensionTbl set the configuration navigation over the configTbl nodes is started. If called with an extensionTbl the navigation is done over that table. The extension table must be built up like in the example, see [7.6.4]. If hide is true nodes that are configured to be hidden will be hidden, see menuItemName[] below.
With downloadCheckOverride it is possible to override the default behavior for package downloading. It can be a function that returns true when download starts and it can be a non-nil value (e.g. false) to block download check totally.

Examples:

```
-- Typical SA usage:
config.setup(nil,nil, function() return fs.stat("/tmp/lock") end)

-- Block download check
config.setup(nil,nil, false)
```

The default function download check function looks like this:

```
local function __dlHandler()
-- default builtin handler
local lock = "/tmp/lock"
if fs.stat(lock) then
   local _,_,hpixels = display.size()
   display.push()
   display.cursor("OFF")
   display.iconInput("OFF")
   display.flush()
   display.anim(hpixels-12,1,1)
   -- add unspecified delay
   system.sound(100,0)
   fs.remove(lock)
   display.pop()
   return true
end
end
```

range[]

Table that specifies valid values for leafs. This table is supported only in TH2. The range is used to validate input data when navigating the setup. The key describes the navigation leaf. Every key in the table has a list which syntax is depending on leaf type. Five different types are supported;

Numerical range:

```
config.range[<table leaf as string>] =
{
min=<minValue>,
max=<maxValue>,
type="number",
unit=<"mm"|"inch"|"dots">,
format={mm=<format>,inch=<format>, dot=<format>},
}
```

Integer range:

```
config.range[<table leaf as string>] =
{
min=<minValue>,
max=<maxValue>,
type="integer",
unit="none"
}
```

Selection range:

```
config.range[<table leaf as string>] =
{
{
<value that will be set to setting>,
<what to show as string in display>
[,{<string shown at selected value>,
   <string shown at non-selected value>}]  -- default is 'radio buttons'
}
, … and so on, defining all choices.
}
```

Numerical pattern range:

```
config.range[<table leaf as string>] =
{
pattern=<numerical pattern complying to rule>,
rule = <string to show as guide>,
limits = {{min=0, max =23},{min=0, max=59},{min=0,max=59}},
len=<number>, -- separators not counted
type="string"
}
```

String range:

```
config.range[<table leaf as string>] =
{
pattern="[0-9a-zA-Z]+", -- Lua pattern
len=<number>,           -- max accepted length
type="string"
}
```

Eg:
```
config.range["configTbl.media.size.length"] =
{min=3,
 max=312,
 type="number",
 unit="mm",
 format={mm="%.3f",inch="%.3f",dot="%d"}
}
config.range["configTbl.system.display.intensity"] =
{min=10,
 max=63,
 type="integer",
 unit="none"}
config.range["configTbl.media.adjustment.mirror"] =
{{false,"NO"},
 {true,"YES"}}
config.range["config._nodes.regional.time"] =
{pattern="%d%d:%d%d:%d%d",
        rule = "HH:MM:SS",
        limits = {{min=0, max =23},{min=0, max=59},{min=0,max=59}},
        len=6, -- separators not counted
        type="string"}
config.range["config._nodes.profile.create"] =
{pattern="[0-9a-zA-Z]+",
 len=8,
 type="string"}
```

editFunc[]
Table that specifies which functionality to call. This table is supported only in TH2. Every key in the table has a list with a function and an argument. The key is the navigation leaf. The function is the function to be called and the argument is the argument to pass. The argument is the global variable that will be edited by the function. The argument must match a 'key' in the range table described above. If the leaf in question is a function it shall not have an item in the editFunc list (see ex: [7.6.4])Four predefined edit functions are available;

```
-- Function to edit a global numerical variable
config.editNumber(<variable as string>
      [,<menu node # as string> -- text for 'F1' key press
      [,<exit>]]) – true, exit on enter else only PgUp
-- Function to select a choice from a radio button list
config.editList(<variable as string>,
            [,<menu node # as string>
            [,<exit>
```

```
            [,radio buttons> -- nil/false use radio buttons
            [,noLeave]]]])   -- true, don't leave on no change.
-- Function list directory (files shown as selections in editList)
-- Nodes using listDir have empty range tables which is populated
-- and filtered before listed.
config.listDir(<variable as string>,
[,<menu node # as string>
[,dir,                          -- directory
[,fnFilter,      -- list filter function
[,radio buttons>]]]])


-- Function to edit a global string variable
config.editString(<variable as string>
            [,<menu node # as string>
            [,<exit>]])
```

When called by the main navigation parser the edit function only receives the first argument, the others are nil. To override a user edit function has to be written and used.

Eg.

```
function noRadioButtons(ts, menuNr)
config.editList(ts, menuNr, nil, false)
end
```

Eg of use:

```
config.editFunc["ut.hello"] = {noRadioButtons, "ut.hello"}
```

enterFunc[]

Table that specifies functions to be called at entry of given nodes. This table is supported only in TH2.

Ex: `config.enterFunc["config._nodes.network"] = myFunc`

This will cause the function `myFunc` to be called (without arguments) every time the node "`config._nodes.network`" is entered.

exitFunc[]

Table that specifies functions to be called at exit of given nodes. This table is supported only in TH2.

Ex:

```
config.exitFunc["config._nodes.network"] =
function()
    -- push settings to LAN and save settings
    configTbl.network.active = true
end
```

This will cause the inline function to be called (without arguments) every time the node "`config._nodes.network` " is exited.

link[]
Table that specifies “links” within the tree. This table is supported only in TH2. This can be used to “reuse” parts of the tree. When entering a node that has an entry in this table, the actual position is set to the value of that entry.

```
config.link[<virtual table node as string>] = <existing table node as string>
```

For instance, WEP settings are the same both in infrastructure and adhoc mode, so the WEP settings node of the infrastructure part is linked to the adhoc part of the tree, like so:

```
config.link["wlan.nodes.mode.infra.wep"] = "wlan.nodes.mode.adhoc.security.wep"
```

menuTitle[]
Table that specifies name to be shown at top of display for that specific table path. This table is supported only in TH2.
Ex: config.`menuTitle [<table node as string>] = <string>`

menuItemName []
Table that specifies menu selection names, order and access rights needed.
Ex: config.

```
menuItemName [<table node as string>] =
{<number>,            -- [1..n] selction number
<string>              -- Selection name
[,<user list>]        -- Optional. List with user rights
-- needed to access this node
[,hidden]] -- true if node to be hidden if hide selected.
```

7.6.4 Network Settings

The network settings are somewhat special. Whereas most settings take effect as soon as they have been set, the network (lan and wlan) nodes need to be activated.
There are a few reasons for this, but the main is speed. Changing the network configuration takes quite some time, and setting everything at once is faster than setting each item individually.

The value of `configTbl.network.active` shows the state of the network settings. If it is `false`, the settings have been changed but not sent to the interface. If it is `true` the current settings are in use, i.e. have been sent to the interface.

Setting `configTbl.network.active` to `true` will push the current settings in `configTbl` to the network interface. Setting it to `false` has no effect.

The preferred use is to call the built-in setup, either by means of the entire default setup, `config.setup()`, or - depending on the requirements - `config.setup("config._nodes.network")`.

7.6.5 Example 1

Application extension to the built in setup with a menu-tree. This is supported only in TH2.

```
--
```

```
-- User addition to SATO setup goes like this
--
-- can be used when a menu is to be displayed.
myTbl = { }
myTbl.number = 25
myTbl.string = "S1"
myTbl.setup = config.setup       -- straight function call


config.range["myTbl.number"] = {min=0, max=99, type="integer", unit="none"}
config.range["myTbl.string"] = {{"S1","String 1"},
                                {"S2","String 2"},
                                {"S3","String 3"}}

config.editFunc["myTbl.number"] = {config.editNumber,"myTbl.number"}
config.editFunc["myTbl.string"] = {config.editList,  "myTbl.string"}

config.menuTitle["myTbl"] =         "APP SETUP"
config.menuItemName["myTbl.number"] ={1,"Number"}
config.menuItemName["myTbl.string"] ={2,"String",{"admin","manager"}}
config.menuItemName["myTbl.setup"] = {3,"Setup"}

config.menuTitle["myTbl.number"] = "NUMBER"
config.menuTitle["myTbl.string"] = "STRING"


-- End of user addition

-- start the setup
config.setup("myTbl")
```

7.6.6 Example 2

Application extension to the built in setup with direct leaf access. This is supported only in TH2.

```
--
-- User addition to SATO setup goes like this
--
-- FOR IMMEDIATE LEAF ACCESS do like this

myTbl = { }
myTbl._number = 25 -- holds number
local leaf = "myTbl._number"
local start = "myTbl.number"
myTbl.number = function(ts,_,exit)
                     return config.editNumber(leaf, _, exit)
               end
config.range[leaf] = {min=0,max=99,type="integer", unit="none"}
config.menuTitle[leaf] = "MY TITLE"
config.setup(start)
print("value after edit:", myTbl._number)
-- cleanup
config.range[leaf] = nil
config.menuTitle[leaf] = nil
```

```
myTbl = nil
```

7.7 Devices

In TH2 the devices possible to use are found in the enumeration section of the configtable, see Configuration ch.[7.6]. Devices are similar to io-streams with one major difference and that is that they don't always contain data. Normal io-streams will block and that's not always desirable. To solve this issue a device has a check received data function. This function can be polled for the number of bytes available in the device stream before calling the blocking function read.

The keyboard device ("/dev/kbd") is reserved by the Keyboard API, see [7.13].

In Linux-based printers there are many different devices, much more complex than what can be described here. The device-functions provides access to the C-level open,read,write,close. Google man 2 open,read,write,close. The deviceObjects can be used with socket.select() to create IO-driven appications.

open()
```
dev_desc = device.open(<path>, <mode>)
```
Device descriptor on success else nil. <path> is a device path (or an unix domain socket) according to the enum section in the Configuration table. <mode> is one of "r" (read), "w" (write) or "rw" (read/write) and can also be complemented with "a" (append) and/or "g" (grab, stops others from using this device). All operations are in binary mode.

close()
```
error = device.close(<dev_desc>)
```
Close a previously opened device <dev_desc>. error is `errno.ESUCCESS` on success else `errno.EPARAMETERS`.

read()
```
string, cnt = device.read(<dev_desc>, <count>)
```
Read up to <count> bytes from the device. string is nil on failure and cnt is an error code. NB! This function is non-blocking in TH2, but blocking in Linux-based printers.

write()
```
cnt, errno = device.write(<dev_desc>, <string>)
```
Write <string> data to device. cnt is nil on failure else number of bytes written. error is `errno.ESUCCESS` on success else `write error number`.

bytesToRead()
```
cnt = device.bytesToRead(<dev_desc>)
```
Get the amount of data available to read from the device. cnt is nil on failure else number of available bytes. NB! In Linux-based printers, not all underlying device drivers support bytesToRead().

Usage:
```
dev_kb = device.open("/dev/kbd", "r")
```

```
repeat
  cnt = device.bytesToRead(dev_kb)
until cnt > 0
str , cnt = device.read(dev_kb, cnt)
device.close(dev_kb)
```

eventHandle()

```
eh = device.eventHandle(<eventGroups>)
```

Creates an special device object that can be used to read score events with. The returned object can be used together with Lua socket.select as well to implement an eventbased AEP application. The events can be filtered by event groups as defined in device.evt.
This is not supported in TH2.

getfd()

```
fd = <deviceObject>:getfd()
```

Used to get the underlying file descriptor to integrate with Lua socket.

x:getEvent()

```
event,eventdata = <eventHandle>:getEvent()
```

Used to read the event and the associated eventdata from a device object created with eventHandle.

```
inotify_event = <inotifyHandle>:getEvent()
```

Used to get a lua table representing the inotify_event:

```
{wd=watch_descriptor,mask=mask,cookie=cookie,name=name*}
```

*) The name attribute is not available for all inotify events.
This is not supported in TH2.

x:getEvents()

```
table_of_events = <eventHandle>:getEvents()
```

Used to read the event and the associated eventdata from a device object created with eventHandle. The return value is a lua table of even length, where `event,eventdata` are available in consecutive pairs at odd and even indicies. The event values are present in require("autoload.evt") indexed by groups. The meaning of eventdata depends on the event.

```
table_of_inotify_events = <inotifyHandle>:getEvents()
```

The function reads all available inotify_events and returns each one in a table. See x:getEvent() for details.

This is not supported in TH2.

reaper()

```
uds = device.reaper(fd)
```

This function is a low-level input event harvester. It creates a C-layer reaper thread that reads independent of lua. It buffers the data read from the device object fd in a UDS. Associated methods are discard() and restore().
This is not supported in TH2.

discard()
```
uds:discard()
```
This method discards new data read by the reaper. It is e.g. used to ignore USB-scanner data when AEP is printing.
This is not supported in TH2.

restore()
```
uds:restore()
```
This method restores collecting new data read by the reaper. It is e.g. used to restore reading USB-scanner data after AEP has printed.
This is not supported in TH2.

sendto()
```
device.sendto(uds_path,message)
```
This method sends the lua string message to the uds_path, specifying the Unix Domain Socket datagram, i.e. the uds_path. It may generate a lua error, in case the message could not be delivered due to data being congested.

This is not supported in TH2.

inotify()
```
inotify_handle = device.inotify()
```
Used to create a inotify handle, which is a Linux io notification system. The inotify_handle is consequently used with:
wd=inotify_handle:add_watch(path,flags)
inotify_handle:rm_watch(watchdescriptor)

The flags supported are enumerated in device.INOTIFY.*:
```
IN_ACCESS, IN_ATTRIB, IN_CLOSE, IN_CLOSE_NOWRITE, IN_CLOSE_WRITE, IN_CREATE,
IN_DELETE, IN_DELETE_SELF, IN_MODIFY, IN_MOVE, IN_MOVED_FROM, IN_MOVED_TO,
IN_MOVE_SELF, IN_OPEN.
```

Read the details on Linux inotify(7) pages, and please make sure to verify the behavior in a printer. Monitoring directories in the printer is somewhat different compared to standard PC:s.
This is not supported in TH2.

```
device.evt
```
This is a table that contains definitions for the eventGroups. To find out about the event-number for individual events see require("autoload.evt")

Example:

```
return (json.encode(device.evt))
{"aep":262144,"statusd":256,"net":8192,"battery":131072,"ifc":1024,"bt":4096,"sb
pl":65536,"sys":32768,"sstatus":16384,"ahd":512,"gui":2048}

evt = require("autoload.evt")
```

This is a table of SATO events that are not described further, but the more interesting ones are found in the subtables listed below.

```
return json.encode(require("autoload.evt"),1,true)
{
 "aep":"table: 0x38e498 /*levels*/",
 "statusd":"table: 0x3983f8 /*levels*/",
 "gui":"table: 0x38e948 /*levels*/"

device.path
```

This is a table of enumerated paths used by AEP. Members are aep_pipe_in, aep_pipe_out (reserved by system), ima and suds (UDS). The UDS (Unix Domain Sockets) are used to send JSON<LF>-messages forwarded to the GUI depending on the context. UDS suds is used by AEP for scanners.

7.8 CLxNX Event system

The CLxNX system propagates events to listeners to distribute changes in mode and settings and similar things. The event mechanism is based on Linux file descriptors.Using the Linux select-API via `socket.select()` provided via LuaSocket in Lua, the application becomes event driven.

```
eventHandle,subscribe=system.callbacks(callbacks [, event_groups[, waitFn]])
```

The callbacks table contains functions to call when events defined in require("autoload.evt") fire. The name of the eventnumber in `evt` must be the name of the callback function. The argument `waitFn` is a wait function that sits and waits for an event on `eventHandle` and that passes on the data to `callbacks.processEvent(event,eventdata)`. The parameter callbacks is modified by the system function, and `callbacks.processEvent(event,eventdata)`, `callbacks.queue(event,fn) , callbacks.run(event,...)` and `callbacks.flat_cb{...}` are populated in the `callbacks` table.
`callbacks` is a required parameter of type table, with keys of type string and values of type function.
`event_groups` is optional. It should be of type table or nil. See more in example below.
`waitFn` is optional. It should be of type function or nil. If it is omitted, a function similar to `eventWait` in the example below is created.

```
-- the name usbInut is "determined by" require("autoload.evt").aep.usbInput
local callbacks = {
  usbInput=
   function(evt,eventdata) -- aep 262152
     --scan for USB keyboard again
```

```
     findScanners()
   end,
}
local eventHandle

local function eventWait()
  local rd,wr,err=socket.select({eventHandle})
  if rd[eventHandle] then
    callbacks.processEvent(eventHandle:getEvent())
  end
end

-- the parameter aep=true comes from usbInput is in the aep table
--(require("autoload.evt").aep.usbInput)
eventHandle=system.callbacks(callbacks,{aep=true}, eventWait)
```

The created tables and functions are

`callbacks.flat_cb{...}`
which is a table indexed by event number and the corresponding handler. These handlers fire every time the event fires.

`callbacks.queue(event,fn[,reg])`
will queue the function `fn` to execute when event fires. The function `fn` must return `true` to fire the next time the event occurs. To run once, return `false` or `nil`.
The reg-parameter can be omitted for events that have a handler. NB! The function will only be fired if the event group is subscribed to.
The function fn is passed the arguments `event,eventdata` from `processEvents`

`callbacks.run(event,...)`
This is the function that handles queued event handlers. It is exposed for override purposes.

`callbacks.processEvent(event,eventdata)`
This is the function that runs eventhandlers in `callbacks.flat_cb` and the queued event handlers using `callbacks.run(...)`.

`events=system.newEvents(init,onerror,onmode)`
Creates an events object with the init-function `init`, the onerror-function `onerror` and the onmode-function `onmode`. The onerror-function is called when an event-function causes a Lua-error. The init-function is called to initialize which objects to hook events to. The onmode-function is called when the events switch mode. The objects can be device descriptors or sockets, aka fd (file descriptors).

The `events` object

`events:init()`
Runs the `init`-function

`t =events:get()`
Returns a table with the current input `fd`:s according to the current mode.

`fd,info =events:get(info)`
Returns the input fd described by info-string

`oldfn,old_info=events:remove(fd)`
Used to remove `fd` from the events list, returns old `fn` and old `info`-string

`oldfn,old_info=events:insert(fd,fn[,sinfo[,mode]])`
Used to add handler `fn` for `fd` with the info-string `sinfo` to be processed in the mode `mode`.
The handler `fn` receives `fd,...` as it's argument, so that the same handler can be used for multiple fd:s. If `mode` is omitted, a default mode is used.

`oldfn,old_info=events:replace(oldfd,newfd,fn[,info])`
Used to replace one entry among the managed handlers.
Fallbacks to `events:insert(newfd,fn,info)`.
NB! `info` is not removed if `oldfd==newfd` if the parameter is omitted.

`events:empty()`
Removes all managed handlers.

`events:drain(fd)`
Drains the unread data in `fd`. Calls `fd.restore(fd,nil)` before performing read until there is not more data to be read.

`events:dispatch(fd,...)`
Calls the registered handler function `fn` for `fd`.

`rd,err=events:wait(timeout)`
Calls `socket.select()`
If timeout is omitted it waits indefinitely.

`err=events:run(timeout)`
Runs one wait and handlers or returns err "timeout" after timeout.
If timeout is omitted, it waits until `events:wait()` returns.
If there are poll handlers registered, events:poll(3) is called.

`err=events:poll_add(fn)`
Adds handler fn to poll at timeout or after event. fn is a function that returns true while it should be kept in the poll loop.

`err=events:poll ([n])`
Calls registered poll handlers at timeout or after event. If n is omitted, it calls them up to 100 times, or until they are finished.

`rv,fd=events:hook_store(name,arg)`

Store a hook arg for later retrieval and use. As an example the commonly running SA stores two patterns named "gui_sa.key" and "gui_sa.key/qty". The parameter arg is a table with the parameters described in hook. If arg is false, the named hook is deleted from the storage. The unhook parameter of "gui_sa.key" unhooks when SA detects Cancel, Page up or Enter. The unhook parameter of "gui_sa.key/qty" unhooks when SA detects Cancel, Page up or Enter or sa.quantity()>0.

```
rv,fd=events:hook(arg)
```

The hook method hooks up a event handler to an event and loops until some exit condition. When that happens, it unhooks the handler and restores it to what it was before. This wraps up common code used to write filters and events in the AEP application SA.

```
arg="named_hook"
arg={
      io=io, -- required string, e.g. "gui","online.io","online:in" in SA
      filter=filter, -- required function
      unhook=unhook, -- optional function when to unhook filter
      port=port, -- optional port name(s).
      loop=loop, -- should the hook()-method loop until unhook:ed?
      timeout=timeout, -- optional timeout (in seconds)
      onevent=onevent -- if timeout is given, onevent is required
    }
```

arg
is either a string referring to a table in hook store or a table with arguments.

io
The io parameter is the named io, and in SA it is usually "online:in", "online:io", "gui"

filter

```
forward=filter(fd,data_in, port_in,...)
```

The filter() returns the data it want to forward to the next handler in the chain. If fd is a producer (e.g. hostdata), data_in should be processed before reading from fd. If filter(...) is the first handler, reading is the normal operation., typical for "online:io", but if filter(...) chains into "online:in", there's normally nothing to read.

unhook

```
unhook(fd,data_in, port_in,...)
```

The function should return true when the filter should unhook itself. The default unhook returns false. When the filter unhooks, the previous handler is restored. It'll also terminate loop.

port
. or * for wildcard or the port names separated with comma, e.g. "1024,9100"

loop
true or loop()

Loop can be `nil,true,false or function`. While it evaluates to true events:hook(arg) calls events:run([timeout]) until unhook:ed.
The loop-exit is possible only when loop-mode is used. Loop-exit will also restore the previous handler.

timeout
onevent(arg,err)
The `timeout` and `onevent` are used together if you need to e.g. send a status or otherwise update without receiving new data.
The `onevent` function is called every time at least one event has occurred. The err-parameter is "timeout" in case of timeout.

It should be noted that it is allowed to have timeouts shorter than 1s, e.g. `timeout=0.1` is approximately 100ms, but too short timeouts will choke the system.

It should be possible to create a series of hooks if anyone would find it useful..

rv The hook method returns a copy of arg that is updated at return. In case runtime errors were experienced in `arg.loop(...)` or `arg.onevent(...)`, `rv.err` is set to the runtime error.

fd The hook method returns the fd it hooks into. Useful for non-looped hooks.

```
modes,explanation_table=events:modeget()
modes=events:modeget(fd)
```
Returns all possible modes and a table with short meaning of the letters if `fd` is omitted.
if `fd` is omitted or the modes where `fd` is enabled. Example:
```
print(events:modeget(),(json.encode(unpack({events:modeget()},2))))
"eiop"     {"e":"error","i":"input","o":"offline","p":print"}

events._cmode
events._defmode
```
Variables that hold the current mode and default mode, "eio".
if `fd` is omitted or the modes where `fd` is enabled. Example:
```
print(events:modeget(),(json.encode(unpack({events:modeget()},2))))
"eiop"     {"e":"error","i":"input","o":"offline","p":print"}

events:modeset(fd,modes)
```
Updates the modes where `fd` is enabled.

```
events:mode (mode)
```
Updates the curent mode to `mode` and triggers `events:onmode(oldmode,newmode)`.

```
t=events:info([fd])
```
Returns `{fn=fn,info=info}` for `fd` or a table indexed by `fd`

Usage example:

```
local events=system.newEvents(
  function(t)
    local toAdd=shallowCopyTbl(usbDevs)
    t:empty()
    for k,v in pairs(toAdd) do
      t:insert(v, scanner, "scanner")
    end
    t:insert(eventHandle, conf.processEvent, "events")
    t:insert(hunt, passDown, "hunt:in")
    t:insert(junk_out, passUp, "hunt:out")
end)

local function eventLoop(timeout, timeoutFn)
  while true do
    collectgarbage("step")
    local err=events:run(timeout)
    if err=="timeout" then
      timeoutFn()
    end
  end
end

eventLoop(10, function() print("nothing happened in 10s") end)
```

7.9 XML

The LXP library using the Expat package is included. It is documented online at

http://matthewwild.co.uk/projects/luaexpat/manual.html

7.10 BIT support

The Lua BitOp module (with additions to be bitlib compatible) is incorporated. It supports the following operations:

cast()

```
num = bit.cast(<num>)
```

This function casts a numeric to a bit compatible.

tobit()

```
num = bit.tobit(<num>)
```

This function normalizes a number to the numeric range of bit operations. Usually this function is not needed since all bit operations use this implicitly.

tohex()

```
num = bit.tohex(<num>[,<n>])
```

This function converts <num> to a hex string with <n> hex digits (default 8). A positive <n> between 1 and 8 generates lowercase hex digits, and a negative <n> generates uppercase hex digits.

bnot()
```
num = bit.bnot(<num>)
```
This function returns the bitwise not of `<num>`.

band(), bor(), bxor()
```
num = bit.band(<num1>[,<num2>...])
num = bit.bor(<num1>[,<num2>...])
num = bit.bxor(<num1>[,<num2>...])
```
These functions returns the bitwise and, bitwise or, or bitwise xor of all of its arguments.

lshift (),rshift (), arshift()
```
num = bit.lshift(<num>,<n>)
num = bit.rshift(<num>,<n>)
num = bit.arshift(<num>,<n>)
```
These functions returns either the bitwise logical left-shift, bitwise logical right-shift, or bitwise aritmethic right-shift of <num> by <n> bits.

rol(), ror()
```
num = bit.rol(<num>,<n>)
num = bit.ror(<num>,<n>)
```
These functions returns either the bitwise left rotation, or the bitwise right rotation of <num> by <n> bits.

bswap()
```
num = bit.bswap(<num>)
```
This function swaps the bytes of <num>, i.e. converts little-endian to big-endian or vice versa.

bits
```
num = bit.bits
```
This property holds the number of bits that are supported.

7.11 Bignum support

Bignum arithmetics is supported. The following operations are available:

new()
```
<bignumObject>,errno = bignum.new(["<value>"[,<base>]])
```
Creates a bignum object. `"<value>"` has to be an integer written as a string, default is "0". `<base>` can be 10 (decimal) or 16 (hexadecimal), default is 10. `errno` is ESUCCESS if OK, else EPARAM or EINVAL. `tostring(<bignumObject>)` returns the value.

base()
```
<base>|errno = <bignumObject>:base([<base>])
```
Get/set base. `<base>` can be 10 (decimal) or 16 (hexadecimal). `errno` is ESUCCESS if OK, else EPARAM or EINVAL.

value()
```
"<value>"|errno = <bignumObject>:value(["<value>"])
```
Get/set value. "`<value>`" is an integer written as a string. `errno` is ESUCCESS if OK, else EPARAM or EINVAL.

neg()
```
boolean|errno = <bignumObject>:neg([boolean])
```
Get/set sign. `errno` is ESUCCESS if OK, else EPARAM.

cmp()
```
<cmpVal>|errno = bignum.cmp(<bignumObjectA>,<bignumObjectB>)
```
Compare two values. The call returns -1 if `<bignumObjectA>` < `<bignumObjectB>`, 0 if `<bignumObjectA>` = `<bignumObjectB>`, and 1 if `<bignumObjectA>` > `<bignumObjectB>`. `errno` is ESUCCESS if OK, else EPARAM.

copy()
```
errno = bignum.copy(<bignumObjectA>,<bignumObjectB>)
```
Make a copy of a bignum object: `<bignumObjectA>` will be a copy of `<bignumObjectB>`. `errno` is ESUCCESS if OK, else EPARAM or EINVAL.

add()
```
errno = bignum.add(<bignumObjectR>,<bignumObjectA>,<bignumObjectB>)
```
Addition: `<bignumObjectR>` = `<bignumObjectA>` + `<bignumObjectB>`. `errno` is ESUCCESS if OK, else EPARAM or EINVAL.

sub()
```
errno = bignum.sub(<bignumObjectR>,<bignumObjectA>,<bignumObjectB>)
```
Subtraction: `<bignumObjectR>` = `<bignumObjectA>` - `<bignumObjectB>`. `errno` is ESUCCESS if OK, else EPARAM or EINVAL.

mul()
```
errno = bignum.mul(<bignumObjectR>,<bignumObjectA>,<bignumObjectB>)
```
Multiplication: `<bignumObjectR>` = `<bignumObjectA>` * `<bignumObjectB>`. `errno` is ESUCCESS if OK, else EPARAM or EINVAL.

div()
```
errno = bignum.div(<bignumObjectQt>,<bignumObjectRm>,<bignumObjectNum>,
                   <bignumObjectDivisor>)
```
Division: `<bignumObjectQt>` = `<bignumObjecNum>` / `<bignumObjectDivisor>`, rounding towards zero, `<bignumObjectRm>` = `<bignumObjecNum>` - `<bignumObjectQt>` * `<bignumObjectDividor>`. `errno` is ESUCCESS if OK, else EPARAM or EINVAL.

pow()
```
errno = bignum.pow(<bignumObjectR>,<bignumObjectA>,<bignumObjectP>)
```

Exponentiation: `<bignumObjectR> = <bignumObjecA> ^ <bignumObjectP>.errno` is ESUCCESS if OK, else EPARAM or EINVAL.

7.12 RFID

No specific RFID interface exists, since there are no RFID-enabled printers using the Lua API.

7.13 TH2 Keyboard and Scanner

The internal keyboard and a connected scanner is accessible though the library keyboard. The keyboard library adds functionality on top of the "/dev/kbd" device (which is reserved by the keyboard library), see ch.[7.7].

7.13.1 Key codes

The internal keyboard keycodes are of the same format as PS/2 keyboard keycodes.

Key	Make code	Key code (returned by keyboard.getEvent())	Special	PS/2 equivalent
Up arrow	0xe0, 0x75	keyboard.kUAm	Yes, UA	U ARROW
Down arrow	0xe0, 0x72	keyboard.kDAm	Yes, DA	D ARROW
Left arrow	0xe0, 0x6b	keyboard.kLAm	Yes, LA	L ARROW
Right arrow	0xe0, 0x74	keyboard.kRAm	Yes, RA	R ARROW
'1'	0x69	keyboard.k1m		KP 1
'2'	0x72	keyboard.k2m		KP 2
'3'	0x7a	keyboard.k3m		KP 3
Page up (Back arrow)	0xe0, 0x7d	keyboard.kPUm	Yes, PU	PG UP
'F1'	0x05	keyboard.kF1m	Yes, F1	F1
'4'	0x6b	keyboard.k4m		KP 4
'5'	0x73	keyboard.k5m		KP 5
'6'	0x74	keyboard.k6m		KP 6
Delete/C	0xe0, 0x71	keyboard.kBSm	Yes, BS	DELETE
'F2'	0x06	keyboard.kF2m	Yes, F2	F2
'7'	0x6c	keyboard.k7m		KP 7
'8'	0x75	keyboard.k8m		KP 8
'9'	0x7d	keyboard.k9m		KP 9
Enter	0x5a	keyboard.kENm	Yes, EN	ENTER
Feed	0x7e	keyboard.kFFm	Yes, FF	SCROLL
Char, 1/a/A	0x12	keyboard.kCHm	Yes, CH	L SHIFT
'0'	0x70	keyboard.k0m		KP 0
'.'	0x71	keyboard.kdm		KP .

The Japan keyboard keycodes are of the same format as PS/2 keyboard keycodes.

Key	Make code	Key code (returned by keyboard.getEvent())	Special	PS/2 equivalent
Up arrow	0xe0, 0x75	keyboard.kUAm	Yes, UA	U ARROW
Down arrow	0xe0, 0x72	keyboard.kDAm	Yes, DA	D ARROW
Left arrow	0xe0, 0x6b	keyboard.kLAm	Yes, LA	L ARROW
Right arrow	0xe0, 0x74	keyboard.kRAm	Yes, RA	R ARROW
'1'	0x69	keyboard.k1m		KP 1
'2'	0x72	keyboard.k2m		KP 2
'3'	0x7a	keyboard.k3m		KP 3
Menu	0xe0, 0x7d	keyboard.kPUm	Yes, PU	PG UP
SHIFT+Up arrow	0x05	keyboard.kF1m	Yes, F1	F1
'4'	0x6b	keyboard.k4m		KP 4
'5'	0x73	keyboard.k5m		KP 5
'6'	0x74	keyboard.k6m		KP 6
Cancel	0xe0, 0x71	keyboard.kBSm	Yes, BS	DELETE
SHIFT+Left arrow	0x06	keyboard.kF2m	Yes, F2	F2
'7'	0x6c	keyboard.k7m		KP 7
'8'	0x75	keyboard.k8m		KP 8
'9'	0x7d	keyboard.k9m		KP 9
Enter	0x5a	keyboard.kENm	Yes, EN	ENTER
Feed	0x7e	keyboard.kFFm	Yes, FF	SCROLL
Char	0x12	keyboard.kCHm	Yes, CH	L SHIFT
'0'	0x70	keyboard.k0m		KP 0
Shift	0x71	keyboard.kdm		KP .
Date	0x03	keyboard.kF5m	Yes,F5	F5
Continue/Pause	0x0d	keyboard.kTBm	Yes,TAB	TAB

7.13.2 Properties

The default translation table (see code table below). The translation table is a Lua table with the keycodes as keys and the values are tables of 3 values. The value tables contains the ASCII key depending on mode [1..3]. Mode can either be digit, upper case or lower case [1|2|3]. When non digit mode is selected every key can map to many characters depending on how many times the key was pressed in a row. This is the same procedure as the mobile phones use to write characters with use of the phone keyboard.

Translation table (default):

```
local td = { '.', '␣€£$¥,.;§@"', '␣€£$¥,.;§@"' }
local t0 = { '0', '+-*/=();<>[]{}^¡|ΘΞΨΩ0', '+-*/=();<>[]{}^¡|ΘΞΨΩ0' }
local t1 = { '1', ".,-?!'%#&¿:;/\¯_()@1" , ".,-?!'%#&¿:;/\¯_()@1" }
local t2 = { '2', 'ABCÅÄÆÀÇ2', 'abcåäæàç2' }
local t3 = { '3', 'DEFÈÉ3ΔΦ', 'defèé3ΔΦ' }
local t4 = { '4', 'GHIÌ4', 'ghiì4' }
local t5 = { '5', 'JKL5Λ', 'jkl5Λ' }
local t6 = { '6', 'MNOÑÖØÒ6', 'mnoñöøò6' }
local t7 = { '7', 'PQRS⌨☞♦', 'pqrs♌7☞♦' }
local t8 = { '8', 'TUVÜÙ8', 'tuvüù8' }
local t9 = { '9', 'WXYZ9', 'wxyz9' }
```

- generates character codes

```
t[string.char(0x69)] = t1
t[string.char(0x6b)] = t4
t[string.char(0x6c)] = t7
t[string.char(0x70)] = t0
t[string.char(0x71)] = td
t[string.char(0x72)] = t2
t[string.char(0x73)] = t5
t[string.char(0x74)] = t6
t[string.char(0x75)] = t8
t[string.char(0x7a)] = t3
t[string.char(0x7d)] = t9
t[string.char(0xf0, 0x69)] = t1
t[string.char(0xf0, 0x6b)] = t4
t[string.char(0xf0, 0x6c)] = t7
t[string.char(0xf0, 0x70)] = t0
t[string.char(0xf0, 0x71)] = td
t[string.char(0xf0, 0x72)] = t2
t[string.char(0xf0, 0x73)] = t5
t[string.char(0xf0, 0x74)] = t6
t[string.char(0xf0, 0x75)] = t8
t[string.char(0xf0, 0x7a)] = t3
t[string.char(0xf0, 0x7d)] = t9

local u = {}
```

- generates for enumerated keys

```
local bs = "BS"
local f1 = "F1"
local f2 = "F2"
```

```
local da = "DA"
local en = "EN"
local fe = "FF"
local la = "LA"
local me = "PU"
local ra = "RA"
local sh = "SH"
local ua = "UA"

u[string.char(0x12)] = sh
u[string.char(0xf0, 0x12)] = sh
u[string.char(0x05)] = f1
u[string.char(0xf0, 0x05)] = f1
u[string.char(0x06)] = f2
u[string.char(0xf0, 0x06)] = f2
u[string.char(0x5a)] = en
u[string.char(0xf0, 0x5a)] = en
u[string.char(0x7e)] = fe
u[string.char(0xf0, 0x7e)] = fe
u[string.char(0xe0, 0x6b)] = la
u[string.char(0xe0, 0xf0, 0x6b)] = la
u[string.char(0xe0, 0x71)] = bs
u[string.char(0xe0, 0xf0, 0x71)] = bs
u[string.char(0xe0, 0x72)] = da
u[string.char(0xe0, 0xf0, 0x72)] = da
u[string.char(0xe0, 0x74)] = ra
u[string.char(0xe0, 0xf0, 0x74)] = ra
u[string.char(0xe0, 0x75)] = ua
u[string.char(0xe0, 0xf0, 0x75)] = ua
u[string.char(0xe0, 0x7d)] = me
u[string.char(0xe0, 0xf0, 0x7d)] = me

keyboard.default = {}
keyboard.default.translator = t
keyboard.default.codepage = "UTF-8"
keyboard.codeTranslator = keyboard.default.translator
keyboard.codeEnums = u
keyboard.codepage = keyboard.default.codepage
```

Keycodes (returned by `keyboard.getEvent()`) may be used instead of the "translated" keys, for a more direct access. They follow the pattern `keyboard.k_m` for the make codes, and `keyboard.k_b` for the break codes. _ is as indicated in the table in [7.13.1].

7.13.3 Methods

getEvent()

```
keyCode, time, breakCode, scanstat = keyboard.getEvent([<scanner>])
```

Read data from the keyboard or scanner. If `scanner` is true, both devices are read from otherwise only the keyboard is regarded. When no data is available `keyCode` is nil. When data is available `keyCode` is one of the keycodes of the keyboard or scanner, `time` is seconds since boot (i.e. os.time() units), and `breakCode` is true if it is a key release. `scanstat` is false if the data comes from the keyboard and a table describing the state of the modifier keys if it comes from the scanner.

keyOptions()

```
options, error = keyboard.keyOptions(<key code>, <mode>)
```

keyOptions() is a help function to return the possible ASCII values of the key pressed depending on mode. `keyOptions()` uses the translation table selected by the property `codeTranslator`.

When there is a match for the specific <key code>, options is a string containing all alternativs and error is `errno.ESUCCESS`. No match sets options to nil and error to `errno.ENOTFOUND`. Mode is the key mode which can be 1 (digit), 2 (upper case) or 3 (lower case).

This function is used to display what alternatives there are for iterative key presses.
This function always converts to UTF-8 encoding. See nativeOptions

Not applicable for key codes from the scanner.

toChar()

```
key, special, error = keyboard.toChar(<keyCode>, <mode>, <iteration>)
```

toChar is a help function to return the actual ASCII value of the key pressed depending on mode and how many times it was pressed in a row. `toChar()` uses the translation table selected by the property `codeTranslator`.
This function always converts to UTF-8 encoding. See nativeToChar.

When there is a match for specific `<keyCode>` key is the ASCII value and error `errno.ESUCCESS`. No match sets key to nil and error to `errno.ENOTFOUND`. `<mode>` is the key mode which can be 1 (digit), 2 (upper case) or 3 (lower case). Iteration is how many times the key has been pressed.

Some keys do not generate ASCII values. Instead they generate enumerated strings. As an example the Left Arrow generates the string "LA". In that case `special` is `true`, otherwise it is `false` (see table in [7.13.1]).

To get the ASCII value from a scanner key code, an additional fourth argument is given -scanstat, the table received from the getEvent function:

```
key, special, error = keyboard.toChar(<keyCode>, <mode>, <_>, <scanstat>)
```

When `<scanstat>` is a table, the third argument is ignored, and `<mode>` is interpreted as a boolean. If `<mode>` evaluates to true, the "special" keys are translated to a two-character string in the same way as the keyboard, otherwise they give either not found or a suitable ASCII character.

Usage (data from both keyboard and scanner):

```
repeat
           keyCode, time, breakCode, sstate = keyboard.getEvent(true)
           if keyCode then
               key, special, error = keyboard.toChar(keyCode, 1, 1, sstate)
    if key then
                                          print(key)
    else
          print(error)
      end
     end
  until keyCode == keyboard.kENb -- Key 'enter' break code
```

nativeOptions()
```
options, error = keyboard.nativeOptions(<key code>, <mode>)
```

This function works as keyOptions, but it returns the data in the native encoding set for the keyboard. This function is needed for e.g. Japanese input to be able to get ShiftJIS-encoded data from the keyboard to pass on to a ShiftJIS-only reader.

nativeToChar()
```
key, special, error = keyboard.toChar(<keyCode>, <mode>, <iteration>)
```

This function works as toChar, but it returns the data in the native encoding set for the keyboard. This function is needed for e.g. Japanese input to be able to get ShiftJIS-encoded data from the keyboard to pass on to a ShiftJIS-only reader.

layout()
```
layout, error = keyboard.layout(<layout>)
Specify keyboard layout to use. Layout can be keyboard.LAY_INTERNATIONAL or
keyboard.LAY_JAPAN. If called without parameters current layout is returned. For
keyboard.LAY_INTERNATIONAL and keyboard.LAY_JAPAN see 7.13.1.
```

7.14 CLxNX Keyboard and Scanner

Connected keyboards can be accessed by using the exkKbd library, which has functions to list, open, read and close external devices.

Get started
```
local extKbd=require("autoload.extKbd")
```

list()
```
keyboards = extKbd.list([filter])
```

This function returns a list with the path and name of all the keyboards and scanners connected to the printer. Its possible to specify a filter for getting only specific devices, this filter is a table of strings that gets matched with the name of the device. There is also a special filter "scanner" that can be used to list only the scanners connected.

open()
device = extKbd.open(path)
Opens a device for reading and returns a descriptor on success else nil. You need to specify a path to the device, this path can be fetched using the above list() function.

grab()
device = extKbd.grab(path)
Same as open() but makes sure no other part of the system can access the device. If a keyboard is opened using grab, then it can not be used to control the rest of the system.

getKey ()
key = extKbd.getKey(device)
Gets a keypress from the device. If "key" is a string, then its a character from a scanner or a keyboard, else if "key" is a number, then its a sytem key e.g. arrows, enter and pageUp. Positive number means key down and negative value means key up.

List of system keys:
8 --Backspace
13 --Enter
112 --F1
113 --F2
120 --F9
13 --NumPadEnter
19 --Pause
33 --PageUp
37 --Left
39 --Right
38 --Up
40 --Down

close ()
extKbd.close(device)
Closes the specified device.

7.15 TH2 Display

The display should be accessible through the library display.

7.15.1 Functions

codepage()
```
codepage|error = display.codepage([codepage])
```
cp is codepage to be used and error is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`. If called without parameters current codepage is returned.

Default:
```
codepage = "UTF-8"
```

```
Codepages:
"DOS-858"
"ISO-8859-1"
"ISO-8859-2"
"ISO-8859-9"
"DOS-737"
"DOS-855"
"DOS-850"
"DOS-852"
"DOS-857"
```

```
"DOS-866"
("Windows-932")
"Windows-1250"
"Windows-1251"
"Windows-1252"
"Windows-1253"
"Windows-1254"
"Windows-1257"
"IBM CP 00869"
"UTF-8"
```

clearText()

```
error = display.clearText()
```

Remove all the text in the display. error is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.

clearBitmap()

```
error = display.clearBitmap([<area>]|[<xPos>, <yPos>, <width>, <height>])
```

This method can take zero, one or four in parameters.
0 - If no in parameters are present, all bitmaps in the display are removed.
1 - area is the area to clear of bitmaps. area can be set to "ICON_AREA" or "TEXT_AREA". "ICON_AREA" clears the icon area, i.e. the two rightmost colums (excluding the icon delimiter) from bitmaps. If the icon delimiter is set, "TEXT_AREA" clears the area not reserved for icons (i.e. the fourteen leftmost columns) from bitmaps. If the icon delimiter is not set, all bitmaps in the display will be removed.
4 - xPos, yPos, width and height defines a rectangle where all bitmaps in the display are removed.
`error` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.

pos()

```
row,col|error = display.pos([row[,col]])
```

row,col are vertical/ horizontal position to set, error is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`. Specific parameter(s) can be left out by writing nil at the parameter position. If called without parameters current row and column are returned.

Default:

```
row = 1
col = 1
```

getText()

```
string, error = display.getText(<row>)
```

Get text on selected row. `string` is `nil` on error and `error` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.

setText()

```
error = display.setText(string[,style])
```

String is the string to display at current cursor position. Cursor position is updated to after the printed string. Strings displayed outside the display area are truncated. Output format is selected by style which can be `"NORMAL"` or `"INVERSE"`. `error` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.

Default:
```
style = "NORMAL"
```

getStringWidth()
```
table,error = display.getStringWidth(<string>)
```
Each character's start column if displayed on the display is represented in the returned table. Characters can have width 1 or 2 depending on the width of the character. `error` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.
The empty string will return the table {1}.

Eg.
```
-- tbl will be {1,2,4,5}
str = "Aあ B"
tbl,err =display.getStringWidth(str)
-- Get length on display
len = tbl[#tbl]-1
-- Set cursor before B
display.pos(1,1)
display.setText(str)
display.pos(1,tbl[3])
```

getTotWidth()
```
width[,error] = display.getTotWidth(<string>)
```
It performs the same job as getStringWidth(), but it only returns the strings total width on the display. The empty string returns 0. In case of errors, `width` will be nil and `error` will indicate what type of error that occurred. In the event of success (normally) `error` is nil.

Eg.
```
str = "Aあ B"
width,err =display.getTotWidth(str)
-- width is 4
```

size()
```
rows, columns, hpixels, vpixels, colFirstRow, error = display.size()
```
Get the amount of character rows, character columns and pixel of the display. `rows, columns, hpixel,vpixel` and `colFirstRow` are set to nil on error and `error` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`. The return-value `colFirstRow` tells how many characters there are on the first row.

cursor()
```
type|error = display.cursor([<type>])
```

Set cursor properties. <type> is one of "OFF" (non-visible cursor), "BLOCK" (=), "BAR" (= |), or "LINE" (= _). error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM. If called without parameters current type is returned.

Default:
type = "OFF"

bitmap()
error = display.bitmap(<path>, <xPos>, <yPos>[,<style>])
Show Windows monochrome BMP file on display. Position is in LCD pixels (top left corner is 1,1). Output format is selected by style which can be "NORMAL" or "INVERSE". error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

Default:
style = "NORMAL"

push()
display.push()
Push entire display state on internal display state stack.

pop()
err = display.pop()
Pop top display state from stack and restore display and state. err is errno.ESUCCESS on ok, else errno.ENOTFOUND.

mode()
mode|err = display.mode([<mode>])
Returns current mode if called without parameters. mode is "AUTO" (default) or "MANUAL". If "AUTO", display is automatically updated after every write to the display. If "MANUAL", manual update must be done by calling display.flush(). err is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

flush()
display.flush()
Force update of the display.

progress()
error = display.progress(<x>, <y>, <width>, <height>, <progress>)
Displays a progress bar with the given coordinates, size and percent complete.
x, y - upper left corner
width, height - width, height in pixels.
progress - percentage (0-100), negative for removal
error - errno.ESUCCESS on success else errno.EPARAM.

anim()

```
error = display.anim(<x>, <y>[, <type>]))
```
Activate animation, remove any existing. x,y - upper left corner, type is either 0 (default) for a 10x10 pixel "dot circle", or 1 for an 11x11 pixel "hour-glass" animation.
Only one animation can be active at a time.

```
errno.ESUCCESS = display.anim(false)
```
Deactivate animation.

```
active, x, y, type = display.anim()
```
Return current status. If `active` is `false`, then `x`, `y`, and `type` are `nil`.

```
status = display.localize()
```
This function will localize the display in regards to number of lines. There is no turning back from this state.

mini()
```
minimode[,error] = display.mini([<bMode>])
```
Get/Set the display in mini-mode.
Returns current state if no arguments are given.
`bMode` is `[false|true]`. When set to `true`, the display switches to a smaller font and can hold 21*6 characters. However, when icons are enabled (iconDelimiter(true)), the first line holds only 14 characters. When mode is switched the bitmap and text layers are cleared.

7.15.1.1 Icon functions

The following functions are used to handle the icons in the display.
As described below, the three lower icons (representing power, wireless, and error) can be set to "auto". This means that the printer firmware will update them automatically. The two upper (representing error and application), however, are always handled by the application software.

iconDelimiter()
```
delimiter|err = display.iconDelimiter([<delimiter>])
```
Returns current value if called without parameter.
delimiter is [`false`|`true`]. When set to true, the line separating the icons from the rest of the display is shown.
When set to false, the line is not shown and the whole width of the display can be used to write text.
`err` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.

iconInput()
```
input|err = display.iconInput([<input>])
```
Returns current value if called without parameter.
input is ["OFF"|"LOWER"|"UPPER"|"`MIXED`"|"NUM"|<path>].

Value	Icon	Description
"OFF"	-	No icon is shown.
"LOWER"		Input lower case letters.

"UPPER"		Input upper case letters.
"MIXED"		Input mixed case words.
"NUM"		Input digits.
<path>	?	The user can display any icon by letting <path> point to a valid bitmap.

`err` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.

iconApplication()

```
application|err = display.iconApplication([<application>])
```

Returns current value if called without parameter.
application is ["OFF"|"JUMP"|"SCROLL"|<path>]
`err` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.

Value	Icon	Description
"OFF"	-	No icon is shown.
"JUMP"		Jump between columns in a table using the arrow keys.
"SCROLL"		Scroll long rows using the arrow keys.
<path>	?	The user can display any icon by letting <path> point to a valid bitmap.

iconError()

```
error|err = display.iconError([<error>])
```

Returns current value if called without parameter.
error is ["OFF"|"AUTO"|"ON"|<path>]
`err` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.

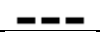
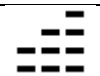
Value	Icon	Description
"OFF"	-	No icon is shown.
"AUTO"	?	The printer firmware will update the icon automatically.
"ON"		Error.
<path>	?	The user can display any icon by letting <path> point to a valid bitmap.

iconLink()

```
link|err = display.iconLink([<link>])
```

Returns current value if called without parameters.
link is ["OFF"|"AUTO"|"0"|"1"|"2"|"3"|"4"|"NOLINK"|"LINK"|<path>]
`err` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.

Value	Icon	Description

"OFF"	-	No icon is shown.
"AUTO"	?	The printer firmware will update the icon automatically.
"0"		No response from card, broken card, card rebooting or installed/removed without reset all. (wlan)
"1"		Not connected. (wlan)
"2"		Weak field strength. (wlan)
"3"		Medium field strength. (wlan)
"4"		Strong field strength. (wlan)
"NOLINK"		Not connected to network or the IP address is 0.0.0.0. (lan)
"LINK"		Connected to network and the IP address is not 0.0.0.0. (lan)
<path>	?	The user can display any icon by letting <path> point to a valid bitmap.

iconWireless()

```
wireless|err = display.iconWireless([<wireless>])
```

This function is replaced by iconLink(). The interface is only kept for legacy reasons and should not be used in new code.

iconPower()

```
power|err = display.iconPower([<power>])
```

Returns current value if called without parameters.
input is ["OFF"|"AUTO"|"AC"|"0"|"1"|"2"|"3"|<path>].
`err` is set to `errno.ESUCCESS` if OK, otherwise `errno.EPARAM`.

Value	Icon	Description
"OFF"	-	No icon is shown.
"AUTO"	?	The printer firmware will update the icon automatically.
"AC"		AC line connected.
"0"		The battery empty.
"1"		The battery is 33% charged.
"2"		The battery is 66% charged.
"3"		The battery is fully charged.
<path>	?	The user can display any icon by letting <path> point to a valid bitmap.

7.16 CLxNX GUI

CLxNX GUI uses a client-server architecture. In order to control the screen and receive user input, an application needs to establish a connection with the GUI server. The GUI server can handle multiple clients, although there's only one LCD to display it on. If multiple browsers are connected to the printer, they are synchronized at gui.send(), gui.push(), gui.pop() but it's not defined which browser will win the gui.push()-race.

The GUI-interface is only available in AEP-mode. AEP-mode is controlled with systemMgmt.setStatus{realmode=systemMgmt.getGuiEnums().status.AEP}.

To interact with the GUI use

local gui= require("autoload.gui").new()

for the object oriented approach (preffered) and

local gui = require("autoload.gui")

to access these low-level and high-level API-calls:

gui.new()
gui.init()
gui.limits()*
gui.connect()*
gui.shutdown()*
gui.send()*
gui.show()*
gui.receive()*
gui.flush()*
gui.push()*
gui.pop()*
gui.run()*
gui.start()*
gui.stop()*
gui.online()*
gui.online_screen()*
gui.onreceive()*
gui.onsend()*
gui.huntd()*
gui.size()*
gui.getfd()
gui. suspend()*

The * denotes the object oriented style can be used.

gui=require("autoload.gui").new()
This creates a self-contained gui object that simplifies the connection management, so that the object can be used with gui:send()/gui:show(), gui:receive(), gui:flush(), gui:online()directly** without establishing the connection and so on.
**) This is a bit misleading when writing an application from scratch. It requires an additional `gui:run(callbacks)` for proper operation. The `callbacks` argument is created with `callbacks={} system.callbacks(callbacks)` as a bare minimum.

socketpath=gui.init()
This function is internally used by the server-side to get the socket path.

{rows=6, offset=9}=gui.size()
{rows=6,select={maxi=5,offset=9}}=gui.limits()
This function reports the number of rows available including prompt/title, and the offset parameter for partial select lists. gui.size() can be ignored.

sock=gui.connect()
gui.shutdown(sock)
gui:connect()
gui:shutdown()
These functions establishes/shuts down a connection with the GUI server.

err=gui.send(sock,msg)
err=gui:send(msg)
err=gui:show(msg)
This function is used to send a message to the GUI server. If err is non-nil, the connection with the GUI server is lost and must be re-established. gui:show(msg) wraps gui:start() gui:send(msg) to simplify usage.

msg,err,ci=gui.receive(sock)
msg,err,ci=gui:receive()
This function is used to receive the message from the GUI server after gui.send(). If err is non-nil, the connection with the GUI server is lost and must be re-established.

sock=gui.flush(sock)
gui:flush()
This function is used to flush any available responses from the GUI server after one ore multiple gui.send().This is useful for displaying progress-bars etc when no responses are expected. If the connection with the GUI-server is lost, nil is returned, and must be re-established.

gui.push()/gui.pop()/gui:push()/gui:pop()
These functions are used in applications that temporarily want to take over the control of the GUI. Used in flows like this:

```
local gui = require("autoload.gui")
local sock = gui.connect()
-- save current GUI state
gui.push()
local msg,r={type="html",content="please wait.."}
gui.send(sock,msg)
if msg.type~="html" then
  r=gui.receive(sock)
else
  -- do something
  -- we don't expect an answer: flush receive buffer
  gui.flush(sock)
end
-- restore previous GUI state
gui.shutdown(sock)
sock=nil
gui.pop()
```

gui.run(callbacks), gui.start(), gui.stop(), gui:run(callbacks), gui:start(), gui:stop(), gui:online(name)
The first call initializes a state machine taking care of switching modes (online|offline|error|printing|AEP-mode). The callbacks parameter passed in is created with system.callbacks(...) and provides system hooks. See the CLxNX event system. When the application enters a point where the GUI-interface is used, gui.start() is called. When leaving the GUI-interface, going back to e.g. ONLINE|PRINTING mode, gui.stop() is called.

gui:online(name) switches to the ONLINE-screen and displays name as protocol name.

mode=gui:online_screen() - returns the current online_screen behavior
gui:online_screen(true|false) - enable|disable the online_screen

gui:onreceive(func(gui,msg,ridMatch) - sets an onreceive function, that is called from gui:receive() when data is received. If the function returns msg, it will be returned from gui:receive(). If the function returns nil, gui:receive() will wait for the next message. The purpose of this is to be able to intercept the normal flow to serve requests initialized from the remote end. Requests initialized from the remote end have reply=t as a query-parameter in HTTP GET.
An example of how this is used for the SA application is for images and table lookups used for screens. The parameters used there are documented here for inspiration and reference. Each request contains an array of either img_request or tbl_request.

```
{img_request [,img_request,...]}
```

where img_request is:

```
{type="img", name="image.png" [,url=1]}
```

When the url-parameter is not nil and not false. the data is wrapped in url(..), suitable for background images. The response per img_request:

```
{type="img", name="image.png", data="<data_url>" [,error="<error message>"]}
```

The response payload is set to max 256kb per request.

```
{tbl_request [,tbl_request,...]}
```

where tbl_request is:

```
{type="tbl", tableName="TableName", columns={"col1","col2",...}, index="col1",
search="<search>", rows=<maxrows> [,offset=<number>][,sortBy="col2"]}
```

The response per tbl_request is:

```
{type="tbl", tableName="TableName", columns={"col1","col2",...}, index="col1",
search="<search>", ,offset=<number>, sortBy="sortBy",colmap={JS-idx-map}
result={{"<col1_data>","<col2_data>",<id1>},{"<col1_data>","<col2_data>",<id2>,}
, ...}
```

The payload for tbl-requests is counted on rows, and is cut off at 512 rows.

7.16.1 Message formats for send and receive

The message formats for send and receive are explained in the following sections. In send a lua-table is passed that selects the type of GUI screen to display. In receive a lua-table is received that responds to what the user did.

7.16.1.1 Common Send Attributes

type = "input" | "select" | "html" | "message"

This is a required attribute, which selects the type of GUI screen to display. Type-specific attributes and responses for the various screen types are described in the following sections.

prompt = <title>

The prompt/title of the screen.

f1 = false | <icon name, as string> | <table>

- When false, disables the left soft button.
- When a string, sets the left soft button icon. It has to be a valid icon key, such as "vk.menu".
- When a table, applies all the attributes in the table to the left soft button. The following attributes have an effect:

- text = <label, as string>
- icon = <icon name, as string>
- enabled = <boolean>

Note: When f1 is disabled, the left soft button no longer responds.

f2 = false | <icon name, as string> | <table>

- When false, disables the right soft button.
- When a string, sets the right soft button icon. It has to be a valid icon key, such as "vk.done".
- When a table, applies all the attributes in the table to the right soft button. The following attributes have an effect:
  - text = <label, as string>
  - icon = <icon name, as string>
  - enabled = <boolean>

Note: When f2 is disabled, the right soft button no longer responds.

sendkeys = true

If specified, all key presses and key releases on any keyboard will be sent back to the client, as keydown, and keyup.

state = <value>

The value of state is sent back to the client in each response, as state=<value>, until the next send. The value can be anything, and of any type.

7.16.1.2 Common Receive Responses

These are common responses.

When the left soft button or F1 is pressed:

{key = "F1"}

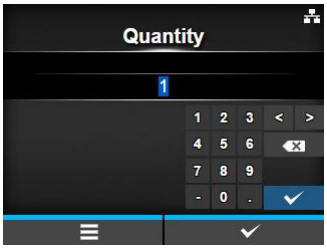
When the Back button or Page Up is long pressed:

{key = "MU"}

When the Back button or Page Up is pressed:

{key = "PU"}

7.16.1.3 Type "input"




```
gui.send(sock, {
  type = "input",
  prompt = "Quantity",
  f1 = "vk.menu",
  value = 1,
  pattern = "%d"
})
```

Prompts the user to input a value.

pattern = <regexp or pattern>

Restricts the input. Only values that match the pattern are accepted by the GUI. Pattern can be a JavaScript regular expression or a pattern of the form SA uses.

pwd = true

If specified, turns input type to password, which hides characters.

keyboard = "default" | "numeric"

If specified, sets the type of virtual keyboard. If omitted, the SA-pattern will select it. If pattern is in Javascript-style, the keyboard is type is default.

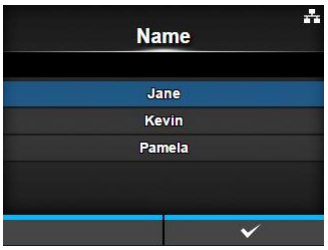
value = <value>

Sets the starting value of the input field.

7.16.1.3.1 Done Response

When Enter or F2 is pressed:
{key = "EN",value = "<value from user input>"}

7.16.1.4 Type "select"




```
gui.send(sock, {
  type = "select",
  prompt = "Name",
  table = {"Jane", "Kevin", "Pamela"}
})
```

Prompts the user to select an item from a list. There's an input-field present where search parameters can be entered to reduce the list.

table = <table of strings>

These are the items that make up the list. Making long lists will reduce responsiveness. Lists shorter than 50 items should be OK. If using larger lists, display a window (partial list).

offset = <non-negative integer>

The index of the first visible list item, or, the viewport offset.

index = <non-negative integer>

The index of the list item to be highlighted, with 0 being the first visible list item. The list item highlighted is, with 0-based offset: offset+index.

pattern = <regex or pattern>

Restricts the input. Only values that match the pattern are accepted by the GUI. Pattern can be a JavaScript regular expression or a pattern of the form SA uses.

input = <value>

Sets the starting value of the input field.

top = true | false
bottom = true | false

Override the default behavior of partial lists. For legacy reasons top is optional, and then offset<9 means that we're at the top of a list, but it is recommended setting top explicitly to control this. For legacy reasons the expression #table<(2*offset+maxi)means that we're at the bottom of a list, but it is recommended setting bottom explicitly to control this.

7.16.1.4.1 Handling Partial Lists

Because applications sometimes deal with databases of several thousands of rows, the select view implements a specific protocol for continuously requesting new data from the application, that is used by SA. For this section sendmsg is used to refer to the message sent to the GUI, and responsemsg refer to the message received from the GUI. The keys to this protocol are sendmsg.offset and the length of sendmsg.table. The code gui.limits().select returns the limits for CLxNX select-view: maxi=5, offset=9. In the following the values 9 and 14 originates from that.
Keep an internal offset, e.g. myOffset, that knows the offset from top.
sendmsg.top=myOffset<=gui.limits().select.offset
sendmsg.bottom=#sendmsg.table<(gui.limits().select.offset*2+gui.limits().select.maxi)

When the GUI detects that the user scrolls upwards and sendmsg.top is false it will request new rows by sending the response {key="UA",index=x}
The application then needs to provide a new sendmsg with table values further up. The new sendmsg.index=0 and the previously highlighted row should be at offset+1. Similarly if the GUI detects that the user scrolls downwards when sendmsg.bottom is false it will request new rows by sending the response {key="DA",index=x}
The application then needs to provide a new sendmsg with table values further down. The new sendmsg.index=gui.limits().select.maxi and the previously highlighted row should be at offset-1.

It's important to note that the application is responsible for handling these responses, probably by sending a new list. No further GUI responses will happen until the application updates the GUI again.

When dealing with partial lists, two additional intermediate responses are sent:
{key="AD"} Sent to go to the bottom of the list and sendmsg.index =gui.limits().select.maxi .
{key="AU"} Sent to go to the top of the list and sendmsg.index=0 .

7.16.1.4.2 Done Response

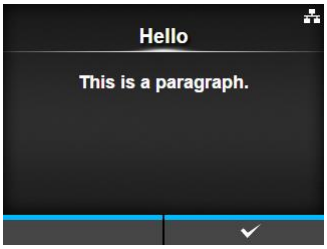
When a list item is selected with Enter or F2:

```
{
  key = "EN",
  input = "<value of input field>",
  value = "<selected item>",*
  index = "<index of selected item>"*
}
```

value is the actual string value of the selected item. index is the position of the selected item, counted from the offset. This means index can have a negative value.

* When Enter or F2 is used with an empty list, value and index are omitted from the response. In this case, it's up to the application to handle an input with no match.

7.16.1.5 Type "html"




```
gui.send(sock, {
  type = "html",
  prompt = "Hello",
  content = "<p>This is a paragraph.<p>"
})
```

A read-only view that displays HTML content.

content = <HTML string>

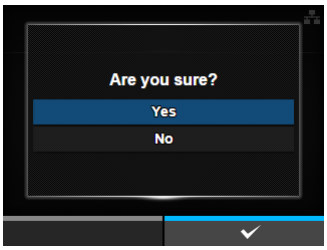
A string of HTML code that will be displayed. If the elements do not fit the allotted space, there will be arrows on the right side indicating that scrolling can be done up or down. Images can be embedded in this style: `<img src="...data.. />`

7.16.1.6 Done Response

When user confirms by pressing Enter or F2:

```
{
  key = "EN"
}
```

7.16.1.7 Type "message"




```
gui.send(sock, {
  type = "message",
```

```
  text = "Are you sure?",
  options = {"Yes", "No"}
})
```

Displays a pop-up with or without a list of options for the user to chose from.

text = <string>

The text to be displayed in the pop-up.

options = <array of strings>

A list of options presented to the user.

allowCancel = true

If set, the user has the option to cancel the pop-up, instead of confirming.

timeout = <seconds>

If set, the pop-up will be automatically confirmed after the specified number of seconds.

7.16.1.8 Done Response

```
{
  accepted = true | false,
  selected = <index> | nil
}
```

If the pop-up was confirmed, accepted will be true, otherwise false. If the pop-up was confirmed, and the options array was provided, selected will be equal to the index of the option that was selected.

7.16.2 Attribute "sendkeys"

When sendkeys = true is specified with gui.send(), every key press that happens on a keyboard is sent to the application, as the attribute:

```
keydown = {
  which = <key code>,
  shiftKey = true | false,
  ctrlKey = true | false,
  altKey = true | false,
  metaKey = true | false
}
```

<key code> is the JavaScript key code for the pressed key. The other attributes indicate whether the different modifier keys were held down when the keyboard event occured.

Similarly, every key release is also sent to the application, as:

```
keyup = {
  which = <key code>,
  shiftKey = true | false,
  ctrlKey = true | false,
  altKey = true | false,
  metaKey = true | false
}
```

Note that the regular responses, described by 5.14.1, are still sent as separate messages in addition to the keydown and keyup messages. It is the responsibility of the application to handle (or ignore) these, and update the GUI with a new screen. Checking for the presence of the `keydown` and `keyup` attributes should suffice.

Warning! The sendkeys-implementation is under construction.

7.17 Database

The internal database format should be accessible through the library sdb.

7.17.1 Functions

connect()

```
status,error = sdb.connect(filename)
```

Connect to database file `<filename>`. `status` is true on success. True is returned if the printer can open the filename for reading, but the actual file contents is not checked for better performance. Subsequent sdb-operations will return errors such as errno.EXMLSCHEMA, if the file format is not recognized.

query(), wquery()

```
table, status = sdb.query(column,searchFor,numRows[,offset])
table, status = sdb.wquery(column,searchFor,numRows[,offset])
```

column is which column to search in, searchFor is the string to search for, numRows is the number of rows to return. The optional parameter offset is 0 by default, but can be used to offset the return values (for scrolling upwards use negative, for scrolling downwards use positive). The wquery method is only guaranteed to support positive offsets.
The searchFor is implemented using strcoll with case-insensitivity. This means the sorting order is language/cultural (locale) dependent. If searchFor is an empty string ("") the first entry in the index will be returned. If no matching is found, the first will also be returned. Retrieving four lines at the end-1 will get the last line.

The default sorting algorithm is string-search, but if the column's sort attribute is set to numeric, numerical sort will be done. The searchFor-comparison operator is 'greater or equal than' with query. The searchFor-comparison operator is 'equal to' with wquery.
The return value table is a table of xml-rows as strings according to our database xml schema.

The wquery function is intended for allWords indices to enable searching of matching words somewhere in the column, and the searchFor parameter is interpreted like this: If it contains a space character, the following non-space character forms an AND expression that must match somewhere on a word-boundary after the initial match.
Example:
The football teams "FC Rubin Kazan", "FC Barcelona", "Liverpool FC" and "FC Internazionale Milano" are stored in the column data in four different rows.

```
sdb.wquery("Team","F",4) -- gets all teams from "FC".
sdb.wquery("Team","R",4) -- gets "FC Rubin Kazan".
sdb.wquery("Team","F K",4) -- gets "FC Rubin Kazan".
sdb.wquery("Team","F I M",4) -- gets "FC Internazionale Milano".
sdb.wquery("Team","LI",4) -- gets "Liverpool FC".
```

If a query is performed on a column while editing the database, an error errno.EBUSY is returned if the index in memory is different from the column queried upon. If the index lookup refers to unexpected data, errno.EREINDEX is returned, to indicate that it is time to reindex the column.

If query or wquery is performed on an empty xml table, a Lua table with no entries is returned. If wquery is performed on an xml table that finds no matches a Lua table with no entries is returned.

The `searchFor` argument is sent as is to the search engine and it is not XML-aware. This means that to be able to search for special characters such as '<','&','>' they need to be encoded to their XML-entity name.

The query operations are affected by localization (l10n) and internationalization (i18n).

lastMatch()

```
table, status = sdb.lastMatch(column,query)
```

This function makes a query, but returns a table containing offset for the maximum offset that returns a result with a matching first word for the query. This is used to get the last match out of a query. Example

```
print(json.encode(sdb.lastmatch("F"))
{offset=2}
```

retreive()

```
table, status = sdb.retrieve(column,id,numRows[,offset])
```

Almost the same as query, but return the first row based on the id-attribute of row. This can be used to get "unsorted rows" when switching from one column to another, but before entering a search term. This behavior is described in ref.[2] section 4.2.4. If retrieve is done on an id that does not

exist, table will be nil and status will be errno.ENOTFOUND. It can also return errno.EREINDEX similar to query().

tquery(),tretrieve(), twquery()

```
table, status = sdb.tquery(column,searchFor,numRows[,offset])
table, status = sdb.tretrieve(column,id,numRows[,offset])
table, status = sdb.twquery(column,searchFor,numRows[,offset])
```

The `tquery, tretrieve` and `twquery` functions are variants of `query, retrieve` and `wquery` that return the xml-data as a table of strings instead of one unparsed string. The columns are indexed from 1 and upwards. The id-attribute from the row tag is stored at index ['id'].
A table column can be sorted by numbers or by string-order.
The `twquery` function is used in SA and for columns ordered by string-order, it will only return rows that match the query as indicated in the table below. If the column is sorted by numbers, it will fallback to `tquery`. The `twquery` function gives additional searching benefits also when an index is created with `allWords` as false for column values containing multiple words.

The `tquery` function return values equal or larger than the given key. When the key is larger than the largest number, it will return not return any row. See below (index,connect left out).

```
SATO>shell.cat("MyTbl.xml")
<?xml version="1.0" encoding="utf-8"?>
<table name="MyTbl" format="format" display="decimal" selectable="true" timestamp="">
<column name="decimal" format="%.0f" sort="numeric" />
<column name="int32" format="%d" sort="numeric" />
<column name="string" format="%s" />
<column name="format" format="%s" />
<row id="1"><c>0.8</c><c>1</c><c>one</c><c>fmt1</c></row>
<row id="2"><c>1.1</c><c>2</c><c>two</c><c>fmt1</c></row>
…
<row id="8"><c>500.11</c><c>80</c><c>eight</c><c>fmt1</c></row>
</table>

SATO>=json.encode(sdb.tquery("string","om",1))
[{"1":"0.8","2":"1","3":"one","4":"fmt1","id":1}]      0
SATO>=json.encode(sdb.twquery("string","om",1))
[]     0
SATO>=json.encode(sdb.twquery("decimal","0.9",1))
[{"1":"1.1","2":"2","3":"two","4":"fmt1","id":2}]      0
SATO>=json.encode(sdb.twquery("int32","0",1))
[{"1":"0.8","2":"1","3":"one","4":"fmt1","id":1}]      0
SATO>=json.encode(sdb.twquery("int32","9999",1))
[]     0
SATO>=json.encode(sdb.tquery("int32","9999",1))
```

[] 0
SATO>=json.encode(sdb.tquery("string","za",1))
[] 0

index()
`numRowsInIndex, status = sdb.index(column[[,reIndex],allWords])`
Create the index for column. The second parameter is false by default. If it is given and true it will start indexing from scratch. If it is false it will update the index, which generally is faster. The third optional argument controls if the indexer treats the column values as having one or many words separated by whitespace that are independantly searchable. The default value for `allWords` is true. The index() function can return errno.EXMLSCHEMA if the xml-data can't be interpreted. The xml-data is expected to be conformant to the tableSchema.xsd.
The return-value `numRowsInIndex` should not be mistaken as the number of rows that are contained in the table. For that see indexInfo(). The return-value shows how many rows in the allocated index that is in use, which for `allWords`-indexes generally is larger.

The allWords-parameter is useful for twquery, but it is not mandatory to improve searching of partial words. See below.

Column value	`allWords`	twquery-key	Found item
SATO Ichiban	`True`	S	*S*ATO Ichiban
SATO Ichiban	`True`	I	SATO *I*chiban
SATO Ichiban	`True`	S I	*S*ATO *I*chiban
SATO Ichiban	`False`	S	*S*ATO Ichiban
SATO Ichiban	`False`	I	<nothing is found>
SATO Ichiban	`False`	S I	*S*ATO *I*chiban

indexInfo()
`tbl = sdb.indexInfo(column)`

If successful it returns a table with info about the index for column. It is possible to iterate over an index by collecting the information provided by this function. See the following example. The table returned contains three members:
`used` - used positions in index
`rows` - unique number of rows
`wType` - type of index, is true if it has indexed multiple words per row.

Example: Iterating over all rows in a table, but see also sdb.traverse!

```
-- Example assumes a table with at least one column:
-- A column called tag and

-- 1st step connect to database for all operations
status, error = sdb.connect('/tmp/table.xml')
status, error = sdb.index('tag')
```

```
local rows = sdb.indexInfo("tag").rows

-- suitable for iterating over many rows
for offs=0,rows-1 do local row = sdb.query("tag","",1,offs) print(row[1]) end

-- retrieving all rows in one go (suitable for less than 100 rows)
local rows = sdb.tquery("tag","",rows,0)
```

newId()
id, error = sdb.newId() - get new id. Each call generates a new id. If editBy has not been called, nil is returned with the error code set.

add()
`status, error = sdb.add(row)` - add row to database containing an id returned from sdb.newId(). The parameter row can be either a table (length 1) or an XML-string.

delete()
`status, error = sdb.delete(row)` - delete row from database. The row can be either a table (length 1) or an XML-string.

change()
`status, error = sdb.change(row)` - change row in database . The row can be either a table (length 1) or an XML-string. NB! If used together with sdb.tquery/sdb.twquery remember that those functions returns a table of rows.

editDone()
`sdb.editDone()` - This is required to complete edit process so that the final XML-data can be written to the database file. There are no return values.

editBy()
`status, error = sdb.editBy(column)` - The edit functions need a proper database index to operate and editBy() selects which index file to use. It does not matter what column data is modified and what column that is used in editBy(), it is needed by the system to lock onto an index until the editing has been completed with editDone().

predict()
`qt, rows = sdb.predict(column,qt,opts,n)` - can be used to build a T9-like search function. As usual column is the column to search in, qt is a table that is passed in and out. It is indexed by integers for results and by the key cache for previous results. The variable opts are the different key options e.g. “abc2” if pressing key 2. The variable n is the number of rows to return. The return values qt holds a query cache, rows is a table with the search results as returned by twquery. This is for TH2 only.

traverse()

`qt, rows = sdb.traverse(column[,queryFunction])` - returns an iterator function that can be used in for loops to iterate through all rows in a table. If large tables are traversed it will be a bit slow.

Example: Iterating over all rows in a table

```
=sdb.connect("/ffs/10.Shoe table.xml")
for k,v in sdb.traverse("ID") do print(k,v.id,v[2]) end
1       2       Adventure
2       3       Stride
3       4       Speed
4       5       Tango
5       6       Climber
6       7       Nature
7       8       Explorer
8       1       Swing
```

cbRegister({cbIndexing=function ,cbError=function})

`status,error = sdb.cbRegister(callbacks)` – register callback functions by passing a table with optional keys cbIndexing and cbError. To unregister callbacks, pass an empty table.

Example: Using cbRegister

```
=sdb.connect("/ffs/10.Shoe table.xml")
=sdb.cbRegister{

cbIndexing=function(start)
if start then print("indexing started") else print("indexing stopped")
end,
cbError = function(operation,error)
print("sdb operation `" .. operation .. "' failed: " .. errno[error])
end
}
```

As indexing can take a long time, it is advisable to show e.g. an animation while indexing. The cbIndexing() callback provides means to do that. When cbIndexing-callback is registered, sdb.traverse, sdb.[t][w]query, sdb.[t]retrieve, sdb.lastMatch and sdb.predict will automatically attempt to create an indexfile if it is missing. NB! The cbIndexing-callback is NOT called if sdb.index() is done manually. cbIndexing is mostly intended for TH2.
The callback cbError() can be used to be notified users and possibly the programmers that an sdb function went bad due to bad arguments or some other fatal error.

push()

`status[, error] = sdb.push()` – push the state of sdb onto the system stack [FILO] for later retrieval (pop). When it is successful it returns ESUCCESS, otherwise nil and error are returned.

pop()

`status[, error] = sdb.pop()` – restores the state of sdb off the system stack [LIFO] for later retrieval (push). When it is successful it returns ESUCCESS, otherwise nil and error are returned.

offset()

`offset[, error] = sdb.offset(column,id) –` retrieves the offset used to get the row with id id. offset is nil and error set (errno.EPARAM, errno.ENOTFOUND) if it fails.

wrap()
`mode = sdb.wrap([mode]) –` get or set the wrap behavior when reaching the end of the table. The legacy mode (`true`) is to wrap to the first row in the index to achieve returning the requested number of rows. If set to `false`, the results from the query methods will be truncated at the final row. The setting is global and remains until restart. The wrap mode setting is saved in `push()` and restored at `pop()`.

limits()
`max_rows_index, min_row_length = sdb.limits(nil|index_rows,nil|min_row_length})`
The limits() function can be used to specify the allocation limits for a sdb database. If nil or 0 is used, the value does not change. The min_row_length parameter will reserve the specified amount of bytes for each table row, but it is not a hard limit.

proc – sdb system variable
`tbl = sdb.proc –` returns the current state of sdb. The contents is changed when the connection or index status changes.

```
sdb.proc.table.props   - attributes from tag "table" (table)
sdb.proc.table.columns – info from colums (by integer and name) (table)
sdb.proc.connectedTo   - path to connected database/table (string)
sdb.proc.activeIndex   - name of current index (column) (string)
sdb.proc.indexFilename – path to current index (string)
sdb.proc.callbacks     - status of registered callbacks (table)
sdb.proc.indexInfo     - info as returned from indexInfo (table)
```

newTable()
`sdb.newTable(path,sdbTable)`
Creates or overwites an sdb XML table in the file `path` with the properties defined in `sdbTable`. `sdbTable.props` sets the table attributes and `sdbTable.columns` the column properties (indexed by integer). The default attribute name for a table will be "Table1"; the default attribute name for a column will be colN, when N is the column order starting from 1.

7.17.1.1 Table property editing

To add and delete table columns, change table properties, and change column properties, three API calls exist.
Not available on TH2.

```
status, error = sdb.addCol(<column_table> (table))
status, error = sdb.deleteCol(<column> (string)[,<pad> (nil,false,number)])
status, error = sdb.changeTable(<table_props> (table), <columns> (table))
status, error = sdb.cleanup([<pad> (nil,false,number)])

status is
- true on success (and error errno.ESUCCESS)
- nil on failure, error set to suitable error number
```

The functions must be called while connected to a database and in edit mode (i.e. editBy() has been successfully called.)

addCol() always adds a column at the last position. All rows get an empty value in the new column (""). <column_table> is a table with the column attributes. At least the 'name' and 'format' fields must be present. addCol will fail if <column_table>.name is an existing column name. addCol does not validate <column_table> fields, except as noted above. Runs an internal cleanup too. Uses the padding (limits) found in the table header.

deleteCol() deletes <column>. deleteCol will fail if called with a non-existing column name. Runs an internal cleanup too. Use the padding (limits) found in the table header (nil), or as provided in the call. When <pad> is false the extra padding is deleted.

changeTable() updates the currently connected database properties with the fields in <table_props> and the column properties with the columns described in <columns>. Any fields not included will cause the existing properties to retain their values. Fields with a non-nil value evaluating to boolean false will remove the corresponding attribute. <columns> is indexed using integers, so if columns 1 and 3 are to be updated, <columns>[2] shall be nil.

cleanup() performs a full database rewrite, removing historic information from the file. It applies padding using the same rules as deleteCol().

The functions are atomic, i.e. changes are written directly to the database table. The editBy/editDone requirement is mostly for API symmetry.

Example

```
sdb.connect(..)
sdb.index(..)
sdb.editBy(..)

-- Add NewColumn last:
sdb.addCol({name="NewColumn", format="%s"})

-- Delete "NewColumn" column
sdb.deleteCol("NewColumn")

-- Update 2nd column:
sdb.changeTable({}, {[2]={name="Second!", format="%d", sort="numeric"}})

-- Make the table selectable, remove gui attribute:
sdb.changeTable({selectable="true", gui=false}, {})

-- Exit edit mode
sdb.editDone()
```

7.17.2 Examples

```
-- Example assumes a table with at least two columns:
-- A numeric ID and a text field Product.
```

```
-- A row like that looks like this:
-- <row id="1"><c>12</c><c>Milk</c></row>
-- The attribute in tag row must be unique

-- 1st step connect to database for all operations
status, error = sdb.connect('/tmp/table.xml')
if not status then abort('Could not connect to database. Error code ' .. error) end

-- 2nd step create/update indexes for query/retrieve/edit operations
status, error = sdb.index('ID')
if not status then abort('Could not index ID. Error code ' .. error) end
status, error = sdb.index('Product')
if not status then abort('Could not index Product. Error code ' .. error) end
-- query from database
t, error = sdb.query('ID', '', 1, 0)
if not t then abort('Could not query database. Error code ' .. error) end

-- t[1] is the XML for the row. Application specific code to modify the columns
row = applicationModifyRowContents(t[1])

-- start an edit session
status, error = sdb.editBy('ID')
if not status then abort('Could not edit database. Error code ' .. error) end

-- in edit session, change row
status, error = sdb.change(row)
if not status then abort('Could not change row. Error code ' .. error) end

-- in edit session, create id for new row
id, error = sdb.newId()
if not id then abort('Could not get new id. Error code ' .. error) end
-- create XML row for this database
newRow = applicationUpdateId(row, id)

-- in edit session, add row
status, error = sdb.add(newRow)

-- in edit session, query/retrieve only work with the editBy column
status, error = sdb.retrieve('ID', id, 1, 0)
if not status then abort('Could not get my new row. Error code ' .. error) end

-- in edit session, query/retrieve by column different from editBy does not work
status, error = sdb.retrieve('Product', id, 1, 0)
if status or error ~= errno.EBUSY then abort('Should be busy editing') end

-- in edit session, end edit session
status, error = sdb.editDone()
if not status then abort('Could not quit editing. Error code ' .. error) end

-- no longer in edit session, query/retrieve works on other columns
-- other existing indexes are updated on query/retrieve
t, error = sdb.retrieve('Product', id, 1, 0)
if not t then abort('Could not retrieve new id from Products. Error code ' .. error) end

-- start new edit session
status, error = sdb.editBy('Product')
if not status then abort('Could not edit database. Error code ' .. error) end

-- in edit session, delete row
status, error = sdb.delete(row)
```

```
if not status then abort('Could not change row. Error code ' .. error) end

-- in edit session, end edit session
status, error = sdb.editDone()
if not status then abort('Could not quit editing. Error code ' .. error) end

-- release all system resources with disconnect
sdb.disconnect()
```

7.17.3 Excel file conversion

XLSXToXML()

`status[,error,rtn3] = sdb.XLSXToXML(pathToXLSX,sheetX,pathToXML[,erange[,cbImport[,bufferSize[,doNotReAlloc[,cacheParser]]]]])`

Converts an Excel sheet to XML. The XML table has to be created first and the number of columns and their names has to match for the conversion to be successful. The first column used in the Excel sheet must be A, and the first row used represents the column names. The sheet name, `sheetX`, has to follow the format “sheet<number>” e.g. “sheet1” even if the sheet has been renamed in the Excel file.

When successful the call returns ESUCCESS, otherwise nil, `error,` and `rtn3. error` can be either EPARAM, ENOENT, ERANGE, ENOMEM, number of columns in the Excel sheet if not the same as in the XML table, or a column name that do not match the XML table. `rtn3` can be either an Excel position (e.g. “A1”) if `error` is ERANGE (which is when a format is wrong in Excel) or ENOMEM (which is when the internal buffer is to small to read all cell data), false if `error` is other errno than ERANGE (i.e. EPARAM or ENOENT), or true if `error` is not an errno number (i.e. number of columns in the Excel sheet if not the same as in the XML table, or a column name that do not match the XML table).

Examples of possible return values:

`errno.ESUCCESS,nil,nil` – No error.

`nil,errno.EPARAM,false` – Parameter error.

`nil,errno.ENOENT,false` – File error.

`nil,errno.ENOMEM,false` – Out of memory.

`nil,errno.ERANGE,”A1”` – Wrong format (in cell A1).

`nil,errno.ENOMEM,”A2”` – Internal buffer is too small (when trying to read cell A2). Allow memory reallococation (default) or increase the buffer size.

`nil,4,true` – Number of columns mismatch (4 columns found).

`nil,”Example_Name”,true` – Column name mismatch (no match for `Example_Name`).

The default behavior for conversion is to convert all strings to fix point numbers; this is useful for looking up EAN-barcodes, but may result in very large numbers that can't be represented (INFINITY). This behavior can be controlled with the `erange` -parameter. To turn off number-conversion all together pass `errno.ESUCCESS`. To turn off when inifinity pass `errno.ERANGE`. The default setting is `nil`.

parameter cbImport

The cbImport-parameter is a lua function that allows for more advanced Excel imports. The signature is this:

```
sdb.cbImport()-result,colCount,filter=cbImport(sdbColumns, xlsxTable)
```

It is called with the sdbColumns defined in sdb and with the columns retrieved from the Excel sheet in xlsxTable. To accept, return `true` and how many columns per row to convert, and a `filter`-function. Otherwise return a string describing the mismatch.

More on the filter()-function

```
write=filter(chunk)
```

The user defined `filter()`-function is returned from `cbImport()`. It is called during the conversion phase to compose a subset of the columns from the Excel sheet and/or a subset of the rows from the Excel sheet. It is called multiple times per row with the raw XML-data. The call patterns for chunk are:

`<row id="n">` - start of a new row. `n` varies and indicates the row count.
`<c>` start of a new column
`</c>` - end of a column
`<c/>` - start and end of a column with empty data ("")
`</row>` - end of a row
`*` - when nothing else is matching, it is the data for the current column. Called 1..n times per column.

By initialing the data at the start of each new row and then at the end of row, return the concatenated string and the row is imported. By returning nil, the row is omitted. By returning a 2:nd return value of true, the import can be terminated in advance.

The parameters bufferSize and doNotRealloc
The bufferSize-parameter (integer) sets the internal buffer size used when reading the sheet files. The doNotRealloc-parameter (boolean) when set to 'true' disables the default memory reallocation when the internal buffer is too small.

cacheParser()

```
cacheParser = sdb.cacheParser(pathToXLSX)
```

This function creates a function closure, i.e. it returns a lua function, to speed up the conversion of the xlx file. It can be used as the cacheParser-parameter in the associated functions.

XLSXHeader()

```
tbl = sdb.XLSXHeader(pathToXLSX,sheetX,cacheParser)
```

This function will return the parsed column names from sheetX in the xlsx-file in a lua table. NB! The first sheet is always called sheet1, regardless of the displayed name in Excel.
Example:

```
> return (json.encode(sdb.XLSXHeader("DateAndFunction.xlsx","sheet1")))
["Date","StrFun","NumFun","Str","Num"]
```

XLSXSheets()

```
tbl = sdb.XLSXSheets(pathToXLSX,makePreview,cacheParser)
```

This function will return an table of sheets and information about them as described below

```
  {
    {
      name=name,      -- sheet name set by user in Excel
      param=sheetN,   -- e.g. "sheet1"
      columns=columns -- e.g. {"preset","Product",..}
      avg=average raw row length
      t75=75% of the rows fit in this size
      max=maximum row length found
      preview={{col1,col2,col3,...},...}
    },
    ...
  }
```

When the makePreview parameter is false, the function doesn't collect preview data (avg,t75,max,preview) from the first 100 rows. The preview function returns at most 10 rows.

importTable()

```
status[,error,rtn3] = sdb.importTable(pathToXLSX,
pathToXML,sheetX[,cachedParser[,cbTypeFn[,erange[,cbImport]]]])
```

This is a wrapper for XLSXToXML() and the return values and parameters are explained there. The process starts by calling XLSXHeader() and then it will call cbTypeFn(sdbtbl) so that the properties of the Table can be specified at import. The columns parameter is a table with properties as below:

```
{
  props={
    name=<string>,
    limits={<maxrows (number)>,<min_row_length (number)>}
  },
  columns={
    {
      name=<string>,
      format=<string>
    },
    …
  }
}
```

7.17.4 sdbObject

The sdbObject takes care of connecting and indexing the sdb table. To use this function, you must load it.

local sdbo=require("autoload.sdbObject").new(path[,column])

The return value sdbo will take of sdb.connect() and the column parameter, so you can leave it out in the shortcut-operations: `tquery, tretrieve, twquery, index, indexInfo, offset, traverse`.

Example:

local sdbo=require("autoload.sdbObject").new("/tmp/Translate.xml","tag")
```
t,e=sdbo:tquery("", 1) -- fetch one row
```

There are additional methods to update the object properties:

`sdbo:setColumn(column)`
Updates the column to use in the shortcut methods.

`sdbo:setImport(cbImport)`
Updates the cbImport-parameter used in XLSXToXML.

`sdb:setNumberRule(rule)`
Updates the number rule (erange) used in XLSXToXML.

XLSXToXML
`status[error,rtn3] =sdbo:XLSXToXML(pathToXLSX[,sheetX])`
This will convert the xlsx-file to the internal sdb-format. The default value for `sheetX` is `'sheet1'`. The parameters from `setNumberRule` and `setImport` will be passed on to the converter.

8

Localization (l10n) and Internationalization (i18n)

These long terms are commonly written as l10n and i18n, because there are 10 letters between l and n in localization and 18 between i and n in internationalization. They help adapting the printer to regional and cultural environments.

They deal with how dates and times are written, how monetary units are written, how numbers are written, how characters are ordered and what messages are used in menus and error messages. The information on how to format the printer-provided locales has mostly been taken from the Unicode Common Locale Data Repository project, http://unicode.org/cldr/

8.1 Loading Localization files

The localization files are loaded with the command loadLocale()

```
status, errorCode = system.loadLocale(dir|file)
```

When there was a problem, nil is returned together with an error code. If the argument passed is a directory the commands try to load: `numeric.lua, monetary.lua, time.lua, collate.def, messages.lua, keyboard.lua and ps2Input.lua`.

If a file is given, it tries to load that file. The result is stored in the global _locale-table, except for the collate.def-data. The current collate-data is exposed as the filename and is found in `_locale.collate`.

8.2 Using localized variants

`system.lnumFormat(number [, numeric])`
Format number as described in table parameter numeric or the global table _locale.numeric.

`system.lmonFormat(number [, monetary])`
Format number as described in table parameter monetary or the global table _locale.monetary, using the local form for the currency.

`system.limonFormat(number [, monetary])`
Format number (monetary) as described in table parameter monetary or the global table _locale.monetary, using the international form for the currency.

`system.ldateFormat(fmt [,time [, timetbl] ])`
Format number (time) as described in table parameter timetbl or the global table _locale.time, using the same rules as the `os.date()`-function. The standard Lua `os.date()` function is not affected by `system.loadLocale()` and adheres to the POSIX locale.

Note! The `os.date()` function only handles one-character specifiers, because of the way Lua is implemented. %-k, %OB, and other two-character specifiers will not be handled according to the following table.
The conversion specifiers supported are:

Specifier	Replaced by	Defined by which `_locale.time` element
`%a`	The locale's abbreviated weekday name	`wday` array ([1] is Sunday, [7] Saturday)
`%A`	The locale's full weekday name	`weekday` array ([1] is Sunday, [7] Saturday)
`%b or %h`	The locale's abbreviated month name	`mon` array ([1] is January, [12] December)
`%B`	The locale's full month name	`month` array ([1] is January, [12] December)
`%OB`	The locale's alternative full month name (usually a capitalized version for beginning of sentences, in case of locales that have lower case names for months)	`alt_month` array ([1] is January, [12] December)
`%c`	The locale's appropriate date and time representation	`c_fmt` string
`%C`	The year divided by 100 and truncated to an integer, as a decimal number [00,99]	
`%d`	The day of the month as a decimal number [01,31]	
`%D`	Equivalent to %m/%d/%y	
`%e`	The day of the month as a decimal number [1,31], a single digit is preceded by a space	
`%E`	Prefix for alternate representations. No alternate representation for %E exists, i.e. %Ex is the same as %x.	
`%F`	Equivalent to %Y-%m-%d (the ISO 8601:2000 standard date format)	
`%g`	The last 2 digits of the week-based year (see note 1 below) as a decimal number [00,99]	
`%G`	The week-based year (see note 1 below) as a decimal number (for example, 1977)	
`%H`	The hour (24-hour clock) as a decimal number [00,23]	
`%I`	The hour (12-hour clock) as a decimal number [01,12]	
`%j`	The day of the year as a decimal number [001,366]	
`%k`	The hour (24-hour clock), formatted with leading space if single digit [0,23].	
`%l`	The hour (12-hour clock), formatted with leading space if single digit [1,12].	

%m	The month as a decimal number [01,12]	
%M	The minute as a decimal number [00,59]	
%n	A newline	
%O	Prefix for alternate representations. Only %OB is available (see above). Others give the same as the original, e.g. %Ox is the same as %x.	
%p	The locale's equivalent of either a.m. or p.m.	am and pm strings
%r	This should not be used, since many countries do not use 12-hour notation. Use %X instead. The time in the locale's a.m. and p.m. (12-hour) notation.	ampm_fmt string
%R	The time in 24-hour notation (%H:%M)	
%s	The time in seconds from EPOCH (1970-01-01 00:00:00). Same value as os.time() returns.	
%S	The second as a decimal number [00,60]	
%t	A tab (ASCII 9), aka \t	
%T	Equivalent to %H:%M:%S	
%u	The weekday as a decimal number [1,7], with 1 representing Monday	
%U	The week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0	
%v	Equivalent to %e-%b-%Y	
%V	The week number (see note 1 below) of the year as a decimal number [01,53].	
%w	The weekday as a decimal number [0,6], with 0 representing Sunday	
%W	The week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0.	
%x	The locale's appropriate date representation	x_fmt string
%X	The locale's appropriate time representation	X_fmt string
%y	The last two digits of the year as a decimal number [00,99]	
%Y	The year as a decimal number	
%z	The offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm).	
%Z	Not supported (normally timezone name or abbreviation).	
%%	The % character	
%-	Prefix for no padding. See note 2 below. E.g. %-m gives "1" in January, instead of "01".	
%_	Prefix for padding with spaces. See note 2 below. E.g. %_m gives " 1" in January, instead of "01".	
%0	Prefix for padding with zeros, only affects %k and %l. E.g. %0k gives "02" for 2 o'clock, instead of " 2".	

Note 1: `%g`, `%G`, and `%V` give values according to the ISO 8601:2000 standard week-based year. In this system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January 1999, `%G` is replaced by 1998 and `%V` is replaced by 53. If December 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday 30th December 1997, `%G` is replaced by 1998 and `%V` is replaced by 01.

Note 2: Padding (the %- and %_ modifiers) affects the following conversions: %C, %d, %e, %g, %G, %H, %I, %j, %m, %M, %S, %U, %V, %W, %y, and %Y.

8.3 Language support in menus

To handle different languages in menus and messages, a translation table is used, that maps words from one language to another. The default translation table is empty and doesn't do anything. Since the firmware always reports its text in English no translation will happen, thus making English the default language.

8.3.1 Translation table

The table to load by the firmware at start is defined by the configuration. If no configuration is found the default is used (English). Translation tables are stored in files under /rom/locales/<language code>. Languages used in several counties may have an added country specifier, if no specifier is used the locale is for the country the language is associated with (Spain for Spanish, UK for English).

Eg.

English (British) translation table: /rom/locales/en.all/messages.lua.
English (American) translation table: /rom/locales/en_US.all/messages.lua.
Swedish translation table: /rom/locales/sv.all/messages.lua.

Additional or changes to current translations can be done by overloading the translation table. By creating a message.lua file according the below example and store it at /ffs/locales/sv.all/messages.lua the default translation table at /rom/locales/sv.all/messages.lua will be replaced at start up. The selection of different languages is just a change of which translation table to load.

Usage example (translation English to Swedish) :

```
_translate = {
  speed = "hastighet",
  pear =  "päron",
  apple = "äpple",
  mt = { __index = function(table, key) return key end }
}
setmetatable(_translate, _translate.mt)
```

8.4 Methods in i18n

new()
i18nObject.new([fromCodepage [,toCodePage]])
Returns a conversion object that holds the fromCodepage and toCodepage parameters. The fromCodepage and toCodepage arguments are strings that describe the codepage as used elsewhere. The default parameter values for fromCodepage and toCodepage are `"UTF-8"`. If only fromCodepage is given, toCodepage will still be "UTF-8".

Windows-932, which is our copy of Shift-JIS, can only be used as from unless to is also Windows-932. The reason for that is that there is no one-to-one-mapping from Shift-JIS to Unicode and from Unicode to Shift-JIS. Windows-932 is not available in all distributions, only the Japanese distribution.

length = <i18nObject>:len(str)
The return value length is a number that is the symbol length of the supplied string.

str = <i18nObject>:sub(str, i [, j])
The return value str is of type string, and it is a substring of the supplied string, using the same logic as string.sub(), but with symbol index instead of byte indexes.

lowercase = <i18nObject>:lower(str)
The return value lowercase is str converted to lower case according to the locale settings. The returned string is converted to the objects toCodePage.

uppercase = <i18nObject>:upper(str)
The return value uppercase is str converted to upper case according to the locale settings. The returned string is converted to the objects toCodePage.

converted = <i18nObject>:conv(str)
The return value converted is of type string and it is str converted from the objects fromCodePage to the objects toCodePage.

value= <i18nObject>:decode(str)
The return value is the unicode number of the passed symbol, or first symbol in the string if it contains multiple. It is decoded using the fromCodepage.

encoded= <i18nObject>:encode(...)
The return value encoded is a string constructed from '...'. '…' is a list of numbers or a table with numbers, that is converted to the objects toCodePage. This can be used to create UTF-8 encoding from a bunch of Unicode numbers, or to encode e.g. Shift-JIS. The numbers are representing Unicode code points or Shift-JIS code points.
Example:

```
local hlp = i18nObject.new() – UTF-8,UTF-8
local utf8bytes = hlp:encode(0x3041,0xFFe5,0x41)
print(utf8bytes:byte(1,-1))
```

```
227 129 129 239 191 165 65
local hlp = i18nObject.new(1252,1252) – UTF-8,UTF-8
local w1252 = hlp:encode(0x20AC,0x41)
print(w1252:byte(1,-1))
128 65
local hlp = i18nObject.new(932,932) – ShiftJIS
local shiftJisBytes = hlp:encode(0x41,0x8a9a)
print(shiftJisBytes:byte(1,-1))
65 138 154
```

i18nStringObject = <i18nObject>:newi18nString(str)
Method to create an object to do string edits on, with respect to the codepage settings of the i18nObject. The argument str can be either nil, string or an i18nStringObject.
Limitation: i18nObject must be symmetric.

8.4.1 Example – delete one symbol at a time from the end

```
helper = i18nObject.new() – convert from defaults of UTF-8
str = "söderifrån"
while(helper:len(str) > 0) do
str = helper:sub(str, 1, -2)
end
```

8.4.2 Example – i18nObject

```
-- convert from Windows 1252 to UTF-8

l2u = i18nObject.new(1252, "UTF-8") -- convert from 1252 to UTF-8
u2u = i18nObject.new() -- convert from/to defaults to UTF-8
l2l = i18nObject.new(1252,1252) -- Convert from/to Windows 1252

str1252 = "söderifrån" -- this string happens to be encoded in 1252.
strutf8 = l2u:conv(str1252)
STR1252 = l2l:upper(str1252)
STRUTF8 = u2u:upper(strutf8)
STRUTF8 = l2u:upper(STR1252) – also can
```

8.5 Methods in i18nStringObject

The methods in i18nStringObject assist in editing input strings in various encodings.

Length = <i18nStringObject>:len()
Return the number of symbols in self.

nrOfBytes = <i18nStringObject>:nrOfBytes()
This method returns the number of bytes to represent self.

<i18nStringObject> = <i18nStringObject>:sub(i [, j])
Returns a new i18nStringObject that contains the substring asked for.

<self>= <i18nStringObject>:insert(string [, i])
Insert string at position i. If i is not given, string is appended at the end of self. If string is an <i18nStringObject> it will be converted to a string.

<self>= <i18nStringObject>:remove(i)
Remove symbol at position i in self.

<self>= <i18nStringObject>:replace(string, i)
Replace symbol at i and insert string at position i. If position i is outside the limits of <self>, string will be appended to <self>. If string is an <i18nStringObject> it will be converted to a string.

start, stop = <i18nStringObject>:find(pattern[[,init],plaintext])
This function works as string.find, but it will return symbol positions instead of "byte" positions.

<i18nStringObject> = <i18nStringObject>:lower()
Returns a new i18nStringObject containing a lower-case representation of self.

<i18nStringObject> = <i18nStringObject>:upper()
Returns a new i18nStringObject containing an upper-case representation of self.

<string> = <i18nStringObject>:str()
Use the str() method to convert the i18nStringObject to to a normal Lua string.

8.5.1 Example using i18nStringObject

```
i18n = i18nObject.new("UTF-8")
i18nString = i18n:newi18nString()
i18nString:insert('o')
i18nString:insert('l',1)
i18nString:insert('l',1)
i18nString:insert('e',1)
i18nString:insert('h',1)

print(i18nString:str() .. " world")
i18nString:replace(i18nString:sub(1,1):upper(),1)
print(i18nString:str() .. " world")
```

-- This example prints
hello world
Hello world

9

Misc functions

9.1 TH2 RTC

```
system.setTime(newSecondsSinceEpoch [, referenceTime])
```

This command writes a new time to the RTC (Real Time Clock) and you give it the number of seconds since the Epoch started, which was 1970-01-01 00:00:00. The second parameter is not supported in TH2. If the second parameter is provided (referenceTime) it is used to calculate how much time has passed since newSecondsSinceEpoch was input. The intended usage is to make it so that when the user sets the time, the RTC is set at the same time, eventhough time passes before the RTC is actually set. When omitted, the current time is used.

9.1.1 Example of setting the RTC

```
system.setTime(os.time({ year = 2008, month = 01, day = 30, hour = 15, min = 20, sec = 25 }))
```

9.2 Buzzer

```
duration|nil,error = system.sound(duration[, frequency [, volume]])
```

The duration parameter is in milliseconds (max 2s), and the frequency parameter is in Hertz (default 400). Due to the limitations of the TH2 hardware, frequency is ignored.

Depending on hardware support the volume can be controlled. By default it is controlled by the system setting configTbl.sys.sound.error, but it can be temporarily overriden by "NONE","LOW","MEDIUM","HIGH".

9.3 Upgrade

```
err[,estr,enumstr] = system.upgrade(<path> [, x[, y, width, height]])
```

Upgrades printer with the provided file; `<path>` may be boot firmware, main firmware, LAN firmware, or a package file. A package file contains a bundle of files; optionally boot, main and/or LAN firmware; Lua code to be executed; as well as a number of user files that will be copied to the printer's file system (anywhere writable, i.e. not /rom).
The optional parameters specify a progress bar, informing a user on the progress of the upgrade. If no optional parameters are given, a progress bar is displayed with the default parameters of x=14, y=52, w=84 (or 100 if `display.iconDelimiter()` is `false`, and h=11. If x is false or explicitly nil, no progress bar will be shown. The progress parameters describe behavior specific for TH2, and are currently ignored on other platforms.

```
remount,items,installs = system.upgrade(<path>, <string-pathscan>)
[err,estr,enumstr] = system.upgrade(<path>, <string-pathscan>)
```

This variant does not upgrade anything. It runs some checks on the package file and returns remount==true if the rootfs needs to be remounted read-write before installing. It returns the number of items inside the package file. It returns installs=true if the pkg-file contains items that changes the firmware. The second argument is a pattern with part:0 or part:1 with 0 for read-only and 1 for read-write , e.g. '/mnt/data:1/:0' .

`err[,estr,enumstr] = system.upgrade(<path>[, <progress-object>])`
This performs firmware upgrade. Sufficient priviliges are expected as well as partitions properly remounted. The 2:nd argument is used to feedback progress on item and item progress. It can be a Lua socket, deviceObject or Lua FILE object.

`err,estr,enumstr = system.upgrade(<number>)`
This invokation returns error information for the given error code. enumstr can be nil.

`err = system.upgrade(true,<path>)`
This invokation requests a sufficiently priviliged process to install the package file. It returns immediately.

The printer should be restarted after a successful firmware upgrade.

9.3.1 Upgrade packages

When upgrading using a package file, only the first matching firmware of each type (boot/ main/LAN) will be upgraded. This means that it is possible to have one package file with boot/main/LAN firmware for several different printers, the ones not matching the current printer will be ignored.
User files that are included have access rights, owner id and a path associated with them. The path may be relative to the current directory at the time of the system.upgrade() call. If a directory component of the path does not exist in the printer, it will be created automatically.
A number of Lua scripts can be included to be executed in the order they are put in the package file. Those are meant for "housekeeping" of the printer, possibly removing old files and directories.
For more information about the package and firmware file formats, see [1].

9.4 Version and information

`tbl, err = system.info()`
Returns table with system information. See example further down.

Returns a table with the some information on the printer and the currently installed system firmware.
The attributes are as follows:
`platform` - string representing the type of printer (`"TH2 "` for Lynx printers).
`model` - string representing a model name of printer (e.g. `"TH208"`).
`version` - version number of the firmware.
`name` - name of firmware.
`boot` and `main` – specific values for the two different kinds of firmware. Each is a table with the following attribute:

`timestamp` The compilation time in `os.time()` units, UTC.

`head` – info about the print head in a table containing:

`width` width in dots

`dpi` resolution in dots per inch.

`dpmm` resolution in dots per mm.

`MAC` - the printer's MAC address, if it has a network interface.

`serial` - the printer's serial number.

`USBSerial` - USB serial number.

`Bluetooth` – information about the Bluetooth module in a table with the following attributes:

`address` MAC address.

`version` Firmware version.

`LANVersion` - the version of the LAN firmware (empty if no LAN option installed).

`LANDate` - the date of the LAN firmware (empty if no LAN option installed).

`WLANVersion` - the version of the WLAN firmware (empty if no WLAN option installed).

`WLANDate` - the date of the WLAN firmware (empty if no WLAN option installed).

WLANSignal - the received signal strength indication, in dBm. -128 is authenticating. 0 is no association or no WLAN option installed.

boardID - the ID of the PCB. Not present if ID register not present.

link – Current network link status.

options – Returns information about the mounted options.

diffFunc – Returns internal information about differences.

MDL – Returns a table with a metatable that access an internal MDL-structure. Also see systemMgmt.getMDL() for how to get a copy of the MDL-struct.

`system.appInfo(<infotbl>)`

Register information about the current application to firmware. Argument is a table with at least the following string attributes:

`name` - name of the application.

`version` - version of the application.

`date` - date of the application.

This information is used in the printer's setup (System/Test/Info) and for TH Works.

When called with no arguments, `system.appInfo()` returns the previously registered table.

`ac, voltage, level, coinOk, powerOff = system.power([refreshCoinState])`

Get power status.

ac is true if printer is externally powered (not using internal battery), false if battery powered.

voltage is the voltage of the battery. 0 if ac is true.

level is the battery level (0 - 3 where 0 if empty and 3 is full). 0 if ac is true.

coinOk is true if the RTC coin battery (CR2032) is functioning. false if it should be replaced.

powerOff is true if the printer is in the process of shutting down.

If the optional parameter (`refreshCoinState`) is set to true, the current coin battery state will be checked. If false or nil, a cached result will be returned.

Note! Version 40.00.02.02 and earlier does not take any parameter. These versions always returns the current coin battery state.

9.4.1 Example

```
t = system.info()
print(t.name .. ", version: " .. t.version .. " on " .. t.platform .. "\nMain
f/w compiled " .. os.date("%F %T", t.main.timestamp) .. "\nBoot compiled " ..
os.date("%F %T", t.boot.timestamp) .. "\nHead " .. t.head.width .. " dots " ..
t.head.dpi .. " dpi " .. t.head.dpmm .. " dpmm\nModel: "..t.model)
```

Could result in the following printout:

```
SATO Lua, version: 40.00.00.00L on TH2
Main f/w compiled 2008-10-03 17:28:12
Boot compiled 2008-10-03 15:00:19
Head 448 dots 203.2 dpi 8 dpmm
Model: TH208
```

```
print((json.encode(system.info()))
```

could result in the following printout:

{"boot":{"timestamp":0},"ntptime":false,"WLANVersion":"","LANDate":"","WLANStatus":"","platform":"armv7l","options":{"ext":true,"parallel":true,"dispenser":false,"cutter":false,"battery":false,"nfc":false,"rfid":false,"bluetooth":true,"rs232":true,"wlan":false,"rtc":true},"head":{"dpmm":8,"dpi":203,"width":104},"WLANSignal":0,"MAC":"C4:ED:BA:8C:4C:E9","USBSerial":"","LANVersion":"3.4.43-WR5.0.1.9_standard","name":"Linux","model":"CL4NX 203dpi","WLANDate":"","main":{"timestamp":0},"serial":"","version":"3.4.43-WR5.0.1.9_standard"}

9.5 Reboot and Shutdown

```
system.reboot([<time>])
```

If an argument is given, the USB connection is stopped and then reboot is delayed until `<time>` seconds has passed. If no argument, the printer reboots immediately. The function does not return.

```
system.shutdown()
```

The printer shuts down, as if the power key had been pressed for 2 seconds. The function does not return.

9.6 Compile

system.compile(src,dst)
Pre-compile the lua file pointed out by src and save it as dst.

9.7 User Access

9.7.1 TH2 only

The printer supports different users, for protection of settings, applications, and other intellectual property. See [1] for a more thorough description of user access/permissions.

`<name>, <err> = system.user([<user | usertbl> [, <password>]])`
Change current user to `<user>`, optionally providing a password (not always needed). Returns `name, errno.ESUCCESS` if `<user>` exists and password is correct or not needed, where `name` is the resulting user name; `nil` and an error number otherwise.
Alternatively, a table of users can be given. The command will go through that table trying the provided password against the user names. Current user (and return value) is set to the first user that the password works for. Note that the current user, as well as the user "user" does not require any password.
If no arguments are given, the return value `name` is the current user name.

`<err> = system.password([<user>,] <oldpassword>, <newpassword>)`
Change password for `<user>`, or current user if not given. `<oldpassword>` must be correct even if current user is allowed to switch to `<user>` without one. `Nil` or an empty string is "no password".

`<tbl> = system.listUsers()`
Returns a table of all users in the printer, ordered so that "user" and the current user come last.

`<err> = system.pushUser()`
`<user>, <err> = system.popUser()`
Pushes or pops the current user. By using this function, an application can restore the current user to a previous one, without providing a password.

9.7.2 Other Printer Models

In FX3 and *NX printers this version of `system.password()` is available, from version 1.10.0 (CLNX), 3.1.0 (PW2NX), and 5.0.1 (FX3).

`<err> = system.password(<user>, <oldpassword>, <newpassword>)`
Change password for `<user>` to `<newpassword>`. `<oldpassword>` must be correct even if current user is allowed to switch to `<user>` without one. Empty string or nil is not allowed.

9.8 Table Serialization

support.readTblXML
support.readTblXML(<path>)
Read back the Lua table in the XML file given in `<path>` (created by `support.writeTblXML`). The table will be recreated in the global scope.

Returns `true` if table could be read, `false` otherwise.

support.writeTblXML
support.writeTblXML(<path>, <table>, <name>)
Write the Lua table, `<table>`, as an XML file, `<path>`. `support.readTblXML` recreates the table in the global variable `<name>`. Circular (self-referencing) tables are allowed. Only table, number, string and boolean values are stored. Only booleans, strings, and numbers may be used as indexes. The XML schema describing the resulting file is located in `/rom/schemas/tblSchema.xsd`.

Returns `true` on success; `false,<errno>` otherwise.

Example:
```
myTbl = {[12]=12, ["12"]="String 12", [true]="true", func=io.write}
support.writeTblXML("t.xml", myTbl, "gTbl") -- write myTbl to file t.xml
```

Contents of `t.xml` (note that `myTbl.func` is not stored):
```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="SATO_LuaTable"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="SATO_LuaTable tblSchema.xsd" name="gTbl">
<value idxtype="string" idx="12" type="string">String 12</value>
<value idxtype="number" idx="12" type="number">12</value>
<value idxtype="boolean" idx="true" type="string">true</value>
</root>
```

`support.writeTbl`
`support.writeTbl(<path>, <table>, <name>)`
Store a Lua table `<table>` as a file (path is given by `<path>`), which when executed (i.e. dofile) recreates the table in the variable named `<name>` in the global scope. Only tables, numbers, strings and booleans are stored. File handles, functions, userdata, and other objects in the table will be set to `nil`. Cyclic (self referencing) tables are not handled (will crash printer).

Returns `true` if successful; `false,errno` otherwise.

Example:
```
myTbl = {[12]=12, ["12"]="String 12", [true]="true", func=io.write}
support.writeTbl("t.lua", myTbl, "tbl") -- write myTbl to file t.lua
tbl = nil
dofile("t.lua") -- recreate the table in "tbl"
```

Contents of tbl.lua (note that `myTbl.func` is not stored):
```
tbl={
["func"]=nil,
["12"]="String 12",
[12]=12,
[true]="true",
}
```

Value of tbl, after `dofile("t.lua")`:
```
tbl = {[12]=12, ["12"]="String 12", [true]="true"}
```

9.9 TH2 Display

support.clearRow()
support.clearRow(<row>)
Clear the specified display row.

support.centerText()
support.centerText(<row>, <string>)
Center the string at requested display row.

9.10 Wait

support.wait()
support.wait(<sec>)
Returns after time elapsed.

9.11 TH2 Password

support.password()
result = support.password(<list>[,< string>[,<prompt>]])
Prompt for password input. Editing follows the rules specified in Navigation, 7.6.3.
list is a list of users that password will be checked against, string is the string to show as title in the display (translated) and prompt is true if password prompting should happen even if current user is within the list. Result will be a string with the entered password if correct and prompt is true, else it will be true|nil depending if password was entered correct or node was left with PgUp.

Eg.
result = support.password({"admin"},"PW",true)

9.12 Command Channel

There is a command channel interface that can be used to transfer files, execute commands, and more, in parallel with Lua execution. For an explanation and specification of this interface and the commands, see [3]. It is possible to control the command channel from Lua, using the API described in this section.

By default, the command interpreter is active, listening to all communication channels except keyboard and scanner interfaces.

`true|nil, <err> = system.ioConnect( { <p1>, <p2>, … } [, <out path>])`
This function selects the I/O devices that shall be read from/written to (i.e. connected to /dev/stdio and possibly monitored for commands). The first argument gives the devices that shall be used for input and shall be a table of device paths. The maximum number of entries depends on printer model. To disconnect from all devices, give an empty table as argument. If at least one of the paths provided can be successfully opened the operation is considered successful.
The second, optional, argument specifies the output path (device or file). If left out, the output follows the input, i.e. if the input comes from /dev/usb, the corresponding output will be sent to /dev/usb as well.
If a device that is already connected is given as an argument, it will be reopened.
If a device is open, but not part of the arguments it will be disconnected.
Returns `true, errno.ESUCCESS` on success and `nil, <err>` on failure.

`{<inpath>}, <outpath>, <error> = system.ioConnect()`
If no arguments are given, the currently connected input devices are returned in a table ({<inpath>}). The table can be empty, but `nil` only on error (`<error>` is then set to the error number). `<outpath>` is the output device path. `<outpath>` is "" (an empty string) if it follows the input or `nil` on error.

`true|false|nil, <err> = system.ioCommands( [ true | false ] )`
Sets the state of the command parsing. `False` turns command parsing off and `true` turns it on. The current (new if argument is given) state is returned. Returns `nil, <err>` on error.

`<event> = system.ioEvent()`
Receive command channel (i.e. file transfer) events. To get information about what has happened, this function is used. The function returns the oldest non-reported (if any) event that has occurred. Events are only returned once. Any arguments are ignored.

`<event>` is an event (a table) or `nil` if no event is available. Events will always have a field called 'type' that describes what type of event it is. Only file transfers to the printer (type="file") are currently logged. File transfer events have one additional field, path, which is the fully qualified path to the transferred file.

Example:
If the files /tmp/newFile.bin and /tmp/evenNewerFile.bin are transferred to the printer, in that order, the values returned from three consecutive calls to system.ioEvent() are:

```
{type = "file", path="/tmp/newFile.bin"}
{type = "file", path="/tmp/evenNewerFile.bin"}
Nil
```

9.13 TH2 Wireless LAN (Wi-Fi, 802.11g)

wlan.getInfo()
Returns a table containing the wireless settings in a more human-friendly form.

Table attributes:
mode "Infrastructure" or "Ad hoc"
SSID The SSID, e.g. "SATO".
channel The selected channel, 1-13.
security "None", "WEP", "Dynamic WEP/LEAP", "Dynamic WEP/TTLS", "Dynamic WEP/TLS", "Dynamic WEP/PEAP", "WPA/PSK", "WPA/LEAP", "WPA/TTLS", "WPA/TLS", "WPA/PEAP", "WPA2/LEAP", "WPA2/TTLS", "WPA2/TLS", or "WPA2/PEAP".

9.14 Standard Stand-alone Application Support

`name, changed = system.linkStandardApp()`
Available in CLxNX and FX3 series printers. Internal function, used in workspaces created by AEP Works. Not intended to be called by end-user created code (scripts, etc).

Shall be called from within an application at startup, current directory is /ffs/apps/X.
It sets up a link (/ffs/apps/sa -> current directory), for backwards compatibility to support hard-coded paths to resources by user scripts.
If sa was a directory, move that to a /ffs/apps/sa_emul directory to preserve that workspace.

Returns the current name (X in /ffs/apps/X/) of the application, and whether or not it changed the link (true/false).

9.15 Bluetooth

This chapter describes the API for the Bluetooth Serial Port Adapter. This is not supported in any printer.
It is only possible to be connected to other Bluetooth SPP (Serial Port Profile) devices. When connecting to devices as Master only one Slave device can be connected at a time.
For more Bluetooth functionality (e.g. change of device name) see chapter 7.6.2.

role()

```
[string,]error = bluetooth.role([“m”| “s”])
```

role() returns, without any argument, the current role of the Bluetooth module; MASTER or SLAVE. With a argument the role of the Bluetooth module will be changed; “m” for master and “s” for slave. error is set to errno.ESUCCESS if OK otherwise errno.EPARAM , errno.EIO, or errno.ENODEV.

scan()

```
table,error = bluetooth.scan([“<max_response>”,“<timeout>”,“<filter>”])
```

scan() returns a table with address, device name, and if the device is paired for Bluetooth devices found in the proximity. If no devices are found the call returns nil. Three argument (all strings) are optional; max_response, timeout, and filter. max_response is the maximum number of devices that can be found (1 ≤ max_response ≤ 8). timeout is the duration that the scan will continue for (scan duration = timeout*1,28sec, 01 ≤ timeout ≤ 30). filter is a hexadecimal number representing Class of Device. Only devices of the Class of Device used as filter will be scanned for (“0000” means all devices). Default values used for these parameters when no argument is given are “8”,”04”, and ”0000”. error is set to errno.ESUCCESS if OK otherwise errno.EPARAM, errno.EIO, or errno.ENODEV.

Ex:
[{"address":"0007BE104016","name":"Datalogic Scanner","paired":true},
{"address":"6C23B9C9F4F7","name":"Xperia X8","paired":true}] 0

connect()

```
error = bluetooth.connect(“<address>”[,“<PIN_code>”])
```

connect() tries to connect as master to a Blutooth device. Arguments are the MAC address of the Bluetooth device and an optional PIN code (if none is given but is required “0000” will be used as default). error is set to errno.ESUCCESS if OK otherwise errno.EPARAM, errno.EIO, errno.ENODEV, or errno.EACCES (incorrect PIN code).

connected()
```
boolean,error = bluetooth.connected()
```
connected() returns if a connection exists or not. error is set to errno.ESUCCESS if OK otherwise errno.EPARAM or errno.ENODEV.

disconnect()
```
error = bluetooth.disconnect()
```
disconnect() will cancel an established connection. error is set to errno.ESUCCESS if OK otherwise errno.EPARAM, errno.ENOTCONN, or errno.ENODEV.

9.16 Calculating hash

sHash=system.cHash(<string>[,<hashing function>])
This function is used to calculate a hash value from a string. The default hashing function is "RIPEMD". It is encoded in Base 64. The other supported hashing function is "ADLER", which returns a hexadecimal string. The TH2 only supports "RIPEMD" and does not accept the second argument.

9.17 ZIP support

This function is used to unzip and read data from a zip file.

9.17.1 Constructor

open()
```
zipObject,error = zip.open(path)
```
open() creates a zipObject and sets the first file in the zip file as the current file.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

9.17.2 Methods

list()
```
table,error = <zipObject>:list()
```
list() returns a table with information about the content of the zip file.
`path` – the directory of a file in the zip file.
`size` – number of bytes used by that file.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.

read()
```
string|number,error = <zipobject>:read(["*a"|"*l"|”*n”|number])
```
read() returns a string or number with data read from the current file. What the call returns depends on the argument(s) which can be combined in any way, e.g. `str1,num,str2,err = <zipobject>:read(CHUNK,”*n”,”*l”)`.
“*a” (or -1) reads the whole file, starting at its current position. If EOF, the call returns an empty string.
“*l” reads the next line without the newline character. If EOF, the call returns nil. This pattern is the default for read().

"*n" reads a number from the file, starting at its current position. If it cannot find a number (not a number at the current position or EOF) the call returns nil.
number reads number bytes from the file, starting at its current position. If EOF, it returns nil, otherwise the call returns a string with at most number characters. read(0) works as a test for EOF, returning nil if EOF or an empty string if there is more to be read.
error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM or errno.EACCES.

setFile()
```
[err,]error = <zipObject>:setFile(path[,password])
```
setFile() sets a new current file in the zip file. A correct password must be given if the file is password protected (supports the Traditional PKWARE Encryption).
error is set to errno.ESUCCESS if OK, otherwise err is set to nil and error is set to errno.EPARAM, errno.ENOENT, or errno.EACCES.

close()
```
[err,]error = <zipObject>:close()
```
close() closes the zip file. error is set to errno.ESUCCESS if OK, otherwise err is set to nil and error is set to errno.EPARAM.

9.18 LED control

The LED-control API is not supported by any printer.
error | <state>, <color> = system.led(<led>[, <state>, <color>])
This function is used to control the leds on the printer. Different leds supports different colors, see table below:

Led	State	Color
"POWER"	"OFF", "ON", "BLINK_SLOW", "BLINK_FAST"	"RED", "GREEN", "ORANGE"
"LINE"	"OFF", "ON", "BLINK_SLOW", "BLINK_FAST"	"GREEN"
"STATUS"	"OFF", "ON", "BLINK_SLOW", "BLINK_FAST"	"RED", "GREEN", "ORANGE"
"LABEL"	"OFF", "ON", "BLINK_SLOW", "BLINK_FAST"	"RED"
"RIBBON"	"OFF", "ON", "BLINK_SLOW", "BLINK_FAST"	"RED"

error is set to errno.ESUCCESS if OK, otherwise errno.EPARAM.
If only the led argument is given, the current state and color of that led is returned.
Not supported on the TH2.

9.19 Autohunter

Autohunter is a daemon interface multiplexer forwarding data between an application and a configurable amount of input ports (max 15) and one output port. Port configuration is handled by system.ahd() API and data exchange is done over a UNIX domain socket.

This functionality is not supported in TH2.

Default values:
Read ports –
{"family":"INET","port":1024,"name":"1024",
"active":true,"tx":0,"rx":0,"connects":0},
{"family":"INET","port":9100,"name":"9100",
"active":true,"tx":0,"rx":0,"connects":0},
{"family":"DEVICE","path":"/dev/g_printer","name":"USB",
"active":true,"tx":0,"rx":0,"connects":0},
{"family":"DEVICE","path":"/dev/ttyO1","name":"tty",
"active":true,"tx":0,"rx":0,"connects":0},
{"family":"IEEE1284","path":"/dev/te6138","name":"IEEE1284",
"active":true,"tx":0,"rx":0,"connects":0},
{"family":"DEVICE","path":"/tmp/btspp_in","name":"BlueTooth",
"active":true,"tx":0,"rx":0,"connects":0},
{"family":"DEVICE","path":"/tmp/spool/lp","name":"LPD",
"active":true,"tx":0,"rx":0,"connects":0},
{"family":"DEVICE","path":"/tmp/spool/ftp/lp/.lp-pipe","name":"FTP",
"active":true,"tx":0,"rx":0,"connects":0},
{"family":"DEVICE","path":"/tmp/ntagi2c_in","name":"NFC",
"active":true,"tx":0,"rx":0,"connects":0}
Write port -
{"family":"DEVICE","path":"/dev/null","name":"null",
"active":true,"tx":0,"rx":0,"connects":0}

Connect to Autohunter

```
local ahd_fd,sPath
sPath=system.ahd().ahd if sPath then ahd_fd=device.open(sPath) end
```

Connect to Autohunter by using the above pattern. If autohunter is disabled, sPath is nil. The SA-application will connect to Autohunter, and it's made available via sa.events (system.newEvents).

Get configuration and active ports

```
tPorts,tWritePort,tCurrentPort=system.ahd([false|true])
tWritePort,tCurrentPort=system.ahd(1)
tCurrentPort=system.ahd(2)
```

The above patterns are used to retrieve port information for all ports, or for the currently used ports. The `tPorts` is a table traversable with ipairs() and if the port name is known it can be indexed by

name. The properties of the current write port is in `tWritePort`. The properties of the current read port is available in `tCurrentPort`. Please note that `tWritePort` and `tCurrentPort` may be `nil`.

Example: Two methods to modify port with name "1024"

```
t=system.ahd() for k,v in ipairs(t) do if v.name="1024" then v.active=false end
end system.ahd(t)
t=system.ahd() t["1024"].active=false system.ahd(t)
```

The port structure
`{name=sname,family=sFamily,port=number|path=sPath,active=false|true}`
A port is defined as a table with the following required keys: family, port|path, name and active. Adding a new TCP/IP-port is done like this:

```
t=system.ahd()
table.insert(t,{name="MyPort",family="INET2",port=1138,active=true})
system.ahd(t)
```

Set read ports and active write port
`true,nil=system.ahd(tReadPorts [,tWritePort])`

To update the read ports and write port pass the updated configuration to `system.ahd()`.
The maximum number of read ports is 15. The attribute `family` defines what kind of category the port belongs to; depending on `family` the attribute `path` or the attribute `port` is required. The attribute `name` is a human readable name of the port.
Using the attribute `active` a port can be activated or deactivated. If the write port's active flag is set tot `false` all output will go to the last used read port.

The port names "1024" and "9100" are reserved and their port numbers are updated following the system settings (configTbl.network.lan.port1, configTbl.network.lan.port3 and configTbl.network.lan.port_queue). To disable follow port, the port name can be changed.

Getting usage statistics
`t=system.ahd(true)`
The autohunter keeps records of the receive `rx` and the transmit `tx` byte counters for each interface and number of `connects` to the interface. The counters can be read or manually set. To set a counter manually include it in the table definition for the interface to `system.ahd()`.

The different families and their description is defined in the table below.

Family	Description
PIPE	A pipe with the name defined by the path will be created and used by the autohunter.

INET	Defines a IPv4/IPv6 socket port to be used. The listen socket is closed when a connection has been accepted.
INET2	As INET, but the listen socket continues to accepts new connections. They are serviced when the previous connection disconnects.
LOCAL	A UNIX domain socket with the name defined by path will be created and used by the autohunter. The listen socket is closed when a connection has been established.
LOCAL2	As LOCAL, but the listen socket accepts new connections also when connection already established.
DEVICE	Ordinary devices like; tty, USB, files etc.
IEEE1284	A special of DEVICE for IEEE1284 handling.
PRINTFILE	The PRINTFILE-family can be used to spool data from a file through the autohunter. The file must be a temporary file (in /tmp/*, /var/volatile/tmp/* or in /mnt/data/user/temp), and it will be deleted after all data has been spooled. The active attribute is set to false when the file has been spooled. The active attribute will also be set to false if the file path is outside the allowed directories.

9.20 Font resources

9.20.1 Resource table

`system.resourceInit()`
Generate the system font information table `system.resource`.
The `system.resource` table contain all bitmap font names, true type font file names and the true type family table.

`system.resource.font` table looks like this;

```
ttFamily ={
  ["AR CrystalMincho-EBGJK"]={
   {
    path="uEBGJK_Min-GDL_Flat.mbf",
    face=1,
    styleName="Regular"
   },
  ..
   {
    path="uEBGJK_Min-GDL_Flat.mbf",
    face=4,
    styleName="Bold Italic"
   },
   Regular={
    face=1,
    path="uEBGJK_Min-GDL_Flat.mbf"
   }
  ..
   ["Bold Italic"]={
```

```
    face=4,
    path="uEBGJK_Min-GDL_Flat.mbf"
   },
  },
 ..
},
bm={"M","OCR-B","POP1","POP2","POP3","PRICE","S","U","X1","X2","X3","XU"},
tt={
   "uEG_Min-GDL_Flat.mbf",
   "SATOSIPLANDLMONOB.ttf",
   ..
   "SATOSIPLGAMMA.ttf",
   "SATOOCRA.ttf"}
}
```

9.20.2 textTTObject extension

By loading the /rom/autoload/system.lua the textTTObject .new, :font and :face method API gets extended to support font family name and style name as defined in the `system.resource.font.ttFamily` table.

```
dofile("/rom/autoload/system.lua")
system.resourceInit()
tt = textTTObject.new("AR CrystalMincho-
EBGJK",nil,nil,nil,nil,nil,nil,nil,nil,nil,"Bold Italic")
```

10

LuaSocket

A Lua extension library named LuaSocket has been included. It adds support for the SMTP, HTTP, and FTP protocols.
For more information on LuaSocket, see [5].

The TCP/IP stack is roughly divided in UDP and TCP (protocols).
UDP - Connectionless data packets (many times used for discovery protocols)
TCP - Connection based, where a connection is established between two parties before sending the data. The data integrity is checksum-checked.

The TH2 firmware does not have full LuaSocket support. The following features have been disabled (applies to Psim and TH2 firmware):
socket.udp - all functions are disabled
socket.tcp - accept, bind, listen, setpeername, and setsockname are disabled.

This means that no UDP based protocols can be written using the TH2 LuaSocket.
This also means that TCP servers cannot be written using the TH2 LuaSocket. This limitation is for TH2 only.

However, the following things can be done (with the proper setup):
DNS-lookup (to translate e.g. www.google.com to an ip address)
HTTP-requests (uses tcp client operations to access webserver)
FTP-requests (uses tcp client operations to access ftp server)
SMTP-requests (uses tcp client operations to send mail)

Example:
This example shows how to fetch information from a web page using LuaSocket.

```
http = require("socket.http") -- Load the HTTP module
text = http.request("http://www.timeanddate.com/worldclock")
```

In CLxNX support for HTTPS was added in "SATO LuaSocket.s 2.0.2", which means the printer can also connect to SSL-encrypted webservers.

```
socket = require("socket") -- Load the socket module
if socket._VERSION:find('%.s') then
  -- this firmware supports HTTPS
  http = require("socket.http") -- Load the HTTP module
  s,h,bt = http.request{url="https://www.google.com"}
end
```

11

lua-websockets

The Lua module lua-websockets has been included. It adds support for WebSocket version 13 conformant clients and servers.
For more information on lua-websockets, see [6].

Clients are available as synchronous and asynchronous (using lua-ev). Servers are available as asynchronous (using lua-ev).

Coroutine based clients and servers (using copas) are not supported.

A simple synchronous client example showing how to create a WebSocket Secure connection:

```
websocket = require("websocket")

-- create a synchronous client and establish a secure connection
client = websocket.client.sync({timeout=1})
client:connect("wss://echo.websocket.org","echo",{mode="client",
protocol="tlsv1_2"})

client:send("some data")
print(client:receive())

client:close()
```

Proxy support has been added to websockets with new method proxy.

```
url, error = client:proxy([url[,clear]])
```

If all parameters left out returns current proxy.
url (string) is proxy url including any credentials (username:password).
clear (boolean) to remove proxy.

```
-- create a synchronous client and establish a secure connection using proxy
cedentials.
client = websocket.client.sync({timeout=1})
client:proxy("https://<username:password>@<proxy ip:proxy port>")
client:connect("wss://echo.websocket.org","echo",{mode="client",
protocol="tlsv1_2"})
```

12

LuaSec

The Lua module LuaSec 0.6 has been included. It adds support for TLS/SSL communication using an already established TCP connection to create a secure session.
For more information on LuaSec, see [7].

A simple client example showing how to create a secure session:

```
require("socket")
require("ssl")

-- TLS/SSL client parameters (at least mode and protocol are required)
local params = {mode="client", protocol="tlsv1_2"}

local conn = socket.tcp()
conn.connect("127.0.0.1", 8888)

-- TLS/SSL initialization
conn = ssl.wrap(conn, params)
conn:dohandshake()
--
print(conn:receive("*l"))
conn:close()
```

Proxy support has been added to LuaSec https. Proxy set same way as for http by using http.PROXY or request parameter proxy. To detect if proxy is supported the following code can be used;

```
local https = require "ssl.https"
local _,c = https.request{url="https://proxy", proxy="https://proxy"}
local httpsProxySupport = c ~= "proxy not supported"
```

Response when using proxy and not might be different when response could have been generated by the proxy server (eg. failure accessing the remote url). Always check response result and code.

For proxy server using credentials and basic authorization username and password is passed in the proxy url;

```
local https = require "ssl.https"
local _,c = https.request({url="<remote site>", sink=ltn12.sink.file(io.stdout),
proxy="https://<username:password>@<proxy ip:proxy port>"})
```

13

lua-ev

The Lua module lua-ev has been included. It adds Lua integration with the event loop libev. For more information on lua-ev, see [8].

How to load the module:

```
local ev = require("ev")
```

14

LuaSQL

The Lua module LuaSQL has been included. LuaSQL is an interface to a DBMS. It enables Lua programs to connect to varies databases including SQLite, ODBC, ADO, Oracle, MySQL, and PostgresSQL. The following paragraph(s) show(s) which databases that are currently supported. For more information on LuaSQL, see [9].

14.1 SQLite (SQLite3)

SQLite is a relational database management system. It implements most of the SQL-92 standard. With LuaSQL and SQLite, it is possible to create and access a SQL database stored in the printer.

Below is an example of how to create and read from a SQLite database on the printer:

```
function CreateDb(dbFile)
  local driver = require('luasql.sqlite3')
  local env = driver.sqlite3()
  local db = env:connect(dbFile)
  db:execute[[CREATE TABLE test(key varchar(50), value varchar(150))]]
  for i=1,100 do
    db:execute(string.format([[INSERT INTO test VALUES ('%d', 'val%05d')]],i,i))
  end
  db:close()
  env:close()
end

function GetFromDb(dbFile, item)
  local ret = "Not found"
  local driver = require('luasql.sqlite3')
  local env = driver.sqlite3()
  local db = env:connect(dbFile)
  local results = db:execute('SELECT * FROM test WHERE key = ("'.. item ..'")')
  local key,value = results:fetch()
  while key do
    if key == item then
      ret = value
    end
    key,value = results:fetch()
  end
  results:close()
  db:close()
  env:close()
  return ret
end
```

```
CreateDb("/tmp/db.sql")
print("23 ->", GetFromDb("/tmp/db.sql", "23"))
```

Otput:

```
23 -> val00023
```

For more information on SQLite, see [10].

15

Cache/Cookie HTTP

The firmware contains the LuaSocket extension and on that a module to handle certain aspects of HTTP requests has been added. It handles session cookies, cache aspects and can create multipart/formdata to make easy implementation of file uploading.

```
local chttp = require("chttp")
```
This line loads the module.

```
r,c,h = chttp.request
```
This function implements socket.http request method but it will refrain from loading the URI if it is marked as cached in the local cache. In the event of a cache hit, c will be 304, from the HTTP/1.1 standard protocol codes "304 Not modified". It is then up to the caller to retrieve from his/her local cache. If a session cookie has been recorded it will be sent along with the request. If the request is POST, it will always be sent to the web server.

```
r,c,h = chttp.clearCache()
```
This function clears the local cache, stored at /ffs/hcache.

```
r,c,h = chttp._upgrade,upgrade()
```
TBD

```
boundary = chttp.formdata.boundary(mix)
headers = chttp.formdata.headers(mix, boundary)
source = chttp.formdata.source(mix,boundary[,sizeit])
```
These functions are used together to create (HTML) formdata for a web server.

```
boundary = chttp.formdata.boundary(mix)
```
This function creates a boundary that is not found in the data that is sent in the request.

```
headers = chttp.formdata.headers(mix, boundary)
```
This function calculates the size of the data and uses the boundary created from the previous function.

```
source = chttp.formdata.source(mix,boundary[,sizeit])
```
This function is a LTN12 source which plugs in nicely with LuaSocket's http client.

`mix` is a table that contains what fields to send in the web form. The contents of mix can be differently detailed depending on need: Example:

```
mix = {
  simple="simple value",
```

```
  anotherSimple={ value=10 },
  fileWithContentType={ filename="/rom/Json.lua",
                        ["content-type"] = "text/plain"},
  fileWithoutContentType={ filename="/rom/Json.lua" }
}
```

The default content-type for files are "application/octet-stream".

A complete example of using together:

```
-- mix defined above
local chttp = require("chttp")
local boundary = chttp.formdata.boundary(mix)
local headers = chttp.formdata.headers(mix, boundary)
r,c,h = chttp.request{url="http://webserver/test.php",
                      method="POST",
                      headers = headers,
                      sink = ltn12.sink.null(),
                      source = chttp.formdata.source(mix,boundary)
                     }
```

More information about how to encode and send HTML forms could be found at
http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.2

16

JSON

JSON, JavaScript Object Notation, is a transport mechanism used to pass data structures between applications. It is used a lot in Web applications, and a native encoder/decoder is available in the API.

The JSON encoder supports the Lua types number, string, table and boolean. Types such as e.g. functions and userdata are ignored. It does not treat Lua nil in any special way, and if you want to encode JSON null you use json.null().

```
obj = json.decode(json_str)
obj,consumed = json.decode(json_str,return_consumed,make_number_index,keep_null)
```

Decodes from the string `json_str` into the Lua object `obj`. Conversion stops when the first object is decoded. JSON arrays with `null` values will be nil, but the index will be incremented. If the parameter `return_consumed` is true, the number of consumed bytes is returned. If `make_number_index` is true, JSON-objects with keys that can be interpreted as integer numbers are converted to number indexes instead of string indexes.
If keep_null is true, JSON-null is encoded to the special json.null, which is encoded to null in json.encode. This can be useful if the Javascript consumer want's to distinguish between undefined and null.
Example:

```
obj = json.decode("[null,2,3]")
for k,v in pairs(obj) do print(k,v) end
-- prints out
2          2
3          3

obj,consumed = json.decode('[null,2,3]{"1":2,"2":4,"mixed":true}')
print(consumed)
-- prints out
10
obj=json.decode(string.sub('[null,2,3]{"1":2,"2":4,"mixed":true}',10+1))
for k,v in pairs(obj) do print(type(k)=="string" and string.format("%q",k) or
k,v) end
-- prints out
"1"        2
"2"        4
"mixed"    true

obj=json.decode(string.sub('[null,2,3]{"1":2,"2":4,"mixed":true}',10+1),nil,true
)
for k,v in pairs(obj) do print(type(k)=="string" and string.format("%q",k) or
k,v) end
```

```
-- prints out
1          2
2          4
mixed      true
for k,v in pairs((json.decode('{"k":null,"v":5}',nil,nil,true))) do
print(type(k)=="string" and string.format("%q",k) or k,v) end
"k"        json.null
"v"        5
for k,v in pairs((json.decode('{"k":null,"v":5}',nil,nil,false))) do
print(type(k)=="string" and string.format("%q",k) or k,v) end
"v"        5
```

`json_str,ignored = json.encode(obj[[,levels|debug][,prettify]])`

Encodes the Lua object `obj` into the JSON string `json_str`. Types that are not encoded in JSON are ignored. The number `ignored` indicates how many objects that have been left out.

Lua tables are encoded as JSON array ([]) if the following Lua expression is true: `(#t > 0) and next(t,#t) == nil`

Otherwise they are encoded as JSON objects ({}).

Cyclic members are printed like this (xxx is the object's address): "table: xxx /*cyclic*/"

If debug is non-false non-serializable lua-functions are added like this:

"name":"function: xxx /*i*/"

If the `debug` argument can be interpreted as a number (e.g. 2 or "1"), it is transformed into how many levels down the object is traversed. This is mostly useful for debugging. If `levels` is `nil`, `false` or `0` the full depth of the object will be encoded.

The third optional argument `prettify` controls if the encoding adds extra whitespace characters and line breaks to make it easier to read the data structure as a human being. When `prettify` is true the table keys are sorted alphabetically.

NB! `json.encode()` will encode the metatable __index if the original table is empty and `__index.__noJSON` is `nil` or `false`.

Example:

```
obj = {json.null(),2,3, function() end}
json_str, ignored = json.encode(obj)
print(json_str,ignored)
-- prints out
[null,2,3]            1
print(json.encode({json.null(),2,3,a=function() end},true,true))
{
 "1":null,
 "2":2,
 "3":3,
 "a":"function: 0x321af8 /*i*/"
}
          1
```

```
t={}
print(json.encode(t))
[]          0
 setmetatable(t,{__index={this_comes_from_the_metatable=true}})
print(json.encode(t))
{"this_comes_from_the_metatable":true}      0
getmetatable(t).__index.__noJSON=true
[]          0
t[1]=1
print(json.encode(t))
[1]         0
```

```
json_null_object = json.null(obj)
```

As Lua nil values cannot be traversed, they are not encoded. Therefore use json.null() in all places where null is required in the receiving end. See example in json.encode.

17

Emulation parsers

The emulation parsers are not supported in any AEP-printer at the moment. To facilitate efficient parsing of datastreams, the parsers library exists. It provides the interface to all supported parsers.

```
parserObject.DEFAULT
```

Constant for defining default callback. See example below.

```
obj = parserObject.newSZPL(fd)
```

Creates a SZPL object. The parameter fd is a file descriptor from which the datastream is read. The file descriptor is created with device.open. The system file descriptor will be closed when obj is garbage collected. An error is thrown if the parameter is wrong.

17.1 SZPL object

All SZPL objects share the same command definitions, but the datastream and caret,tilde and delimiter are stored inside the object. When initialized, defaults are set. The single byte commands (STX,ETX,SI) are internally mapped to their long forms.
After creating the first SZPL object, the command definitions need to be registered to the parser, see obj.register() below.

```
obj.register(tcmds)
```

The register-function is used define commands and their parameters together with the Lua callback function. A Lua error is thrown if parameter errors are discovered. The parameter `tcmds` is a Lua table and the keys define the commands and the value, which is another table, defines properties for

the command. Example:

```
tcmds =     {
  ["^FO"]={cb=zpl.XA,argc=2},
  ["^CC"]={cb=zpl.CC,bytes=1},
  ["~CC"]={cb=zpl.CC,bytes=1},
  ["~FD"]={cb=zpl.CC,argc=1, trim=false, cati=true},
  ["~DG"]={cb=zpl.DG,argc=3, self=true},
  [parserObject.DEFAULT]=zpl.commandNotHandled,
}
```

The table attributes recognised for the callbacks are:

Name	Default	Description
cb		Lua callback function. Attribute is mandatory
argc	nil	Argument count. Values between 1-15 are accepted, but not in combination with bytes.
bytes	nil	Parser reads the following bytes as argument. Values between 1-15 are accepted, but not in combination with argc.
cati	nil	Command breaks argument only on caret or tilde if set to true
self	nil	If self is true, the parser object is passed as second argument to callback.
trim	true	Arguments are trimmed from leading and trailing whitespace. This parameter needs to be set to false for ^FD. When this parameter is false, spaces are kept, but ctrl are stripped off from leading,trailing and middle.

All SZPL-commands are expected to be three bytes, except for the ^A-command. The byte following A is put as first argument. The commands given should always be registered with the default settings for caret and tilde. The index `[parserObject.DEFAULT]` is used as the fallback for when the parser does not recognise a command.
The parser will call the callbacks on successful completion of the command specification. The signature of the callback functions are all the same:

```
callback(tArgs[, parserObject])
```

No callback returns any value. The first argument passed to the callback is a Lua table and tArgs[1],… are the given arguments. If a parameter gets an empty value, the value is nil. Thus, if no value is given for an argument, the parameter, e.g. tArgs[2], is nil. The callback function does not return any value. The second argument `parserObject` is passed only to callbacks registered with `self=true` and to `parserObject.DEFAULT`.

```
toldRegistration = obj.override[cmd]
obj.override[cmd] = tnewRegistration
```

Callback override can be done by accessing the override-table. A table read returns the old callback in a table with the same attributes as in register. A table set changes the cmd callback. It is possible

to set a command that previously has no callback, but doing so to register many callbacks is inefficient and register should be used instead.

```
status,ecode,estr = obj:parse(tstorage)
```

The parse method reads the datastream and parses the data. When a complete command including arguments have been parsed, it calls the registered callback with tstorage as argument, and if registered to with obj as well. The passed argument must be a Lua table that does not go out of scope between repetitive calls to parse. When the callback returns, parsing continues. When the whole datastream is parsed, the parse method returns, so a typical usage of the parse method looks like this:

```
local tstorage = {}
repeat
-- call other functions here:
  -- e.g. keyboard scanning
local status, ecode, estr = obj:parse(tstorage)
if not status and ecode then
    print("Lua runtime error in parser callback:",ecode,estr)
else if not status then
    print("EOF on datasource")
    break
end
until forever
```

Usage of the Lua table passed to parse (here tstorage) is during the lifespan of the parser object reserved for use by the parser object. When the file descriptor points to an input device data comes into the printer in chunks that cause the parser to return between the chunks. If tstorage is examined at a time like that, it may contain a few arguments that will be passed to the callback function first when all arguments are received. The status code is `errno.EAGAIN` when the data source is low on data.

```
caret,delimiter,tilde = obj:cdt()
obj:cdt(caret,delimiter,tilde)
```

Method to get/set caret, delimiter and tilde of the SZPL object. The parameters must be strings with one character, or nil. If nil the current value is maintained. If no arguments are passed, the current values are returned.

```
status, ecode = obj:db(path, orientation, maxHeight, maxWidth, top2baseline,
                       spaceWidth, numberOfChars, copyrightOwner)
```

Download (~DB) bitmap font, `path, orientation, copyrightOwner` are strings and the others are numbers. If db fails, status is nil.

```
status, ecode = obj:dg(path, total, width)
```

Download (~DG) graphics, `path` is string and the others are numbers. If dg fails, status is nil.

```
status, ecode = obj:du(path, total)
```

Download (~DU) graphics, `path` is string and `total` is number. If du fails, status is nil.

```
status, ecode = obj:dy(path, fileFormat, fileExtension, totSize, bytesPerRow)
```

Download (~DY) graphics, `totSize` and `bytesPerRow` are numbers. The others are strings. If dy fails, status is nil.

```
get=obj:fh()
obj:fh(set)
```

The fh method allows getting/setting the hex indicator. This is required if it is a control character.

```
local arg = {}
status = obj:param(arg,[untilCond])
```

The param method allows retrieving data from the datasource for parsing special commands.

When `untilCond` is nil `param` reads the datastream and returns a parameter according to the current set caret/delimiter/tilde. When all the commands parameters have been read, `status` is nil. This means the caret or tilde is the next byte in the buffer. When a complete parameter is parsed `status` is `errno.ESUCCESS`. When no data is available in the data source `errno.EAGAIN` is returned, but low data is a normal condition. If the data source is a file and EOF is reached, the status will still be `errno.EAGAIN`. Typical usage is like below:

```
function cbVarArgs(fix,parser)
local s
repeat
    repeat
      local arg = {}
      s = parser:param(arg)
    until not s or s == errno.ESUCCESS

    if s == errno.ESUCCESS then
      print("arg is:", arg[1])
    end
until not s or s ~= errno.ESUCCESS
end
```

The method may generate a Lua error, but it is caught by method parse.

The argument `untilCond` can be either a number (>=0) or a string containing stop characters. The request is fulfilled when `nil` is returned.
The data is returned as string-chunks in a table, and the chunk-size depends on many different reasons; the data needs to be concatenated to get the full string.
If 0, the parser will return the next byte in the buffer without consuming it so that the Lua program can give up parsing if an end-condition is met.
When a string is passed, all characters in it are interpreted as stop characters. Any of those will stop further reading. To get passed it, call `parser:param(tArg,1)`.

```
-- peeking (looking at byte without consuming it)
```

```
local s
local data = {}
repeat
    s = parser:param(byte,0)
until not s
if data[1] == '_' then
-- discard _
parser:param(data,1) -- we know _ is received
-- read 2 bytes
data = {}
repeat
    s = parser:param(data,2)
until not s
-- data may be split in several sub-strings
data = table.concat(data)
-- convert to hex number
data = tonumber(data,16)
print("hex-code converted to:", data)
else
-- read until any of the current caret,delim,tilde
local stoppers = {parser:cdt()}
stoppers = table.concat(stoppers)
data = {} -- reset
repeat
    s = parser:param(data,stoppers)
until not s
-- data may be split in several sub-strings
data = table.concat(table)
-- peek stopper byte
local peek = {}
parser:param(peek, 0) -- we know stopper already received
peek = peek[1]
end
```

18

System Management

The system management library is not present in TH2.

```
name = systemMgmt.aepTxtName()
err = systemMgmt.aepTxtName(name)
```

Used to get/set the title-attribute in the GUI settings for configTbl.aep.txt.

```
rng = systemMgmt.aepTxtRng()
err = systemMgmt.aepTxtRng(rng)
```

Used to get/set the configRange["configTbl.aep.txt "]. This is used to create some dynamics in the settings tree. The `rng` is expected to be executable lua.

```
systemMgmt.convertLvlToVolt()
systemMgmt.convertVoltToLvl()
```

Internal functions for displaying sensor readings correctly.

```
tInfo | nil,error = systemMgmt.getAppsInfo()
```

Returns a table with application info from registered applications.
{name="name",version="version"}

```
tEnums = systemMgmt.getGuiEnums()
```

Returns a table of tables with enumerations that the GUI uses so that it can base its code on variable names instead of hardcoded constants. It contains mappings from number to string too.
Some examples:
tEnums.guiWarningEnum:{"1":"ribbonNearEnd","2":"paperNearEnd","4":"receiveBufferNearFull","ribbonNearEnd":1,"16":"headError","32":"changePlaten","64":"cleanHead","128":"changeTPH","256":"changeCutter","512":"calendarError","1024":"unknownHead","2048":"changeBattery1","4096":"changeBattery2","cleanHead":64,"changePlaten":32,"paperNearEnd":2,"receiveBufferNearFull":4,"changeTPH":128,"calendarError":512,"changeBattery2":4096,"headError":16,"changeCutter":256,"commandError":8,"changeBattery1":2048,"unknownHead":1024,"8":"commandError"}

tEnums.guiStateEnum":{"1":"GUI_custom","2":"GUI_online","3":"GUI_offline","4":"GUI_error","5":"GUI_settings","6":"GUI_helps","7":"GUI_init","GUI_online":2,"GUI_error":4,"GUI_custom":1,"GUI_init":7,"GUI_settings":5,"0":"GUI_unknown","GUI_helps":6,"GUI_unknown":0,"GUI_offline":3}

tEnums.guiNotifyEnum:{"1":"waitExtIo","2":"waitLblTaken","4":"usbConnected","waitLblTaken":2,"generic":8,"usbConnected":4,"waitExtIo":1,"rfidEnabled":16,"16":"rfidEnabled","8":"generic"}

tEnums.status:{"1":"ONLINE","2":"OFFLINE","3":"ERROR","4":"POWERDOWN","5":"INIT","6":"UPGRADING","7":"POWERSAVE","8":"WAKEUP","UPGRADING":6,"OFFLINE":2,"INIT":5,"POWERSAVE":7,"ERROR":3,"WAKEUP":8,"ONLINE":1,"POWERDOWN":4}

```
path = systemMgmt.getHexDumpPath()
```
Returns the path used for hexdump and buffer dump files.

```
tbl = systemMgmt.getMDL()
```
Returns a full copy of the MDL-structure. If you want the value for x it is more efficient to use system.info().MDL.x

```
systemMgmt.getNewResolution()
```
Internal functions for getting info about newly detected print head resolution.

```
systemMgmt.getRfidErrorDisplay()
```
Internal function that controls displaying RETRY softkey.

```
tSoftkeys = systemMgmt.getSoftkeys()
Returns a table describing the soft-key definitions in ONLINE, OFFLINE and
ERROR-states.
{{{state=0,name="left",code=0,nx=false,confirm=false},
  {state=0,name="right",code=0,nx=false,confirm=false}},
   ...
   vkey_LEFT=1, vkey_RIGHT=2, OFFLINE=2,ONLINE=1,ERROR=3
}

A softkey is defined by
{

   state=state,        -- read-only
   name="string",      -- name written at soft-key in GUI. Some names have
                       -- special meanings and are converted to icons
                       -- vk.cancel,vk.apply,vk.feed
   code=n,             -- number used for key-code when pressed.
                       -- Must be positive and < 32768.
                       -- 0 is used to disable soft-key
  nx=true|false,       -- translate name (false)
  confirm=true|false,  -- display a confirm-box before
                       -- actually submitting keycode to application
}
```

```
tStatus|nil,error = systemMgmt.getStatus([selectors])
```
Returns a table describing the system status.
Example:
> =(json.encode(systemMgmt.getStatus()))
{"pendingReboot":false,"series":0,"warningStates":0,"netLink":true,"lcount":0,"guiState":0,"hexDumpMode":0,"motion":false,"notifyStates":0,"btLink":0,"tlcount":0,"guiError":0,"processing":false,"mode":5,"wifiStrength":0,"errText":"","tlcountOffset":0, "runState":0}

Name	Description

pendingReboot	True when a setting/condition has changed/239luetoot that requires a system reboot.
warningStates	A bit pattern of active warnings in the printer. 0 means no warnings are active. The bits are enumerated i `guiWarningEnum`.
Lcount, tlcount tlcountOffset	Label count of labels pending to be printed, total count printed since power on, total label count offset for GUI.
guiState	Reflects what the GUI shows (`guiStateEnum`)
notifyStates	A bit pattern similar to `warningStates`, enumerated in `guiNotifyEnum`.
Motion	True when the motors are moving
netLink	True when the printer believes it has 239luetoot link
btLink	Reflects the 239luetooth state.
guiError	The error code that caused the error state. The symbolic name can be retrieved by `errno[num]`. The English text can be retrieved with `errno.text(code)`
processing	True when the printer is processing data or printing.
Mode	The printer status/state enumerated in `status`.
wifiStrength	The signal strength for Wi-Fi.
hexDumpMode	A data-dump mode enumerated in `hexDumpEnum`.
runState	Bit field describing reasons why the printer shall stay awake (i.e. what blocks sleep mode and auto power off). systemMgmt.getGuiEnums().runningEvent describes the bits.

By passing in selectors flexible return values are possible.

```
selectorBased = systemMgmt.getStatus([selectors])
```

The selectors can be strings, .e.g `"guiError","btLink"` to return the two parameters. The selectors can be tables, e.g. `{"guiError","btLink"}` to return a table with the two properties.

```
tUsbInfo = systemMgmt.getUsbInfo()
```

Returns Plug'n'Play info for USB.

Example:

return (json.encode(systemMgmt.getUsbInfo()))

{"bcdDevice":534,"iProduct":"SATO CL4NX 203dpi","iManufacturer":"SATO CORPORATION","idVendor":"0x0828","iPNPstring":"MFG:SATO;CMD:PCL,MPL;MDL:CL4NX 203dpi ; ","idProduct":"0x0122"}

The interface from aep to set the system status.

```
ESUCCESS|nil,error,errstr = systemMgmt.getWifiStatus()
```

Returns WiFi-information.

Example:

```
> return (json.encode(systemMgmt.getWifiStatus()))
{"p2p":{"group":{"ifname":"","key":"","role":""},"connections":[],"ifstat":{"state":"","ssid":"","channel":0,"p2p_device_address":"","bssid":"","freq":0,"ipv4":"0.0.0.0"},"signal":{"speed":0,"rssi":0,"power":-32768}},"wlan":{"ifstat":{"state":"COMPLETED","ssid":"STLE","channel":11,"p2p_device_address":"aa:bb:cc:dd:ee:ff","bssid":"ff:ee:ee:dd:88:99","freq":2462,"ipv4":"123.1.2.3"},"signal":{"speed":54,"rssi":-57,"power":14}}}
```

```
percentage,item,itemCount= systemMgmt.guiProgress()
```

```
percentage,item,itemCount= systemMgmt.guiProgress(percentage[,item[,itemCount]])
```
Get/Set progress-information about percentage completed, item number and total item count.
```
tInfo = systemMgmt.guiProgress({[activate=true|false],[ktitle="translate-
key"],[filename="filename"]})
```
Get/Set progress-information about activation, ktitle and filename. ktitle is localized.

```
state,err = systemMgmt.logEvents([true|false])
```

Get/Set if score events are logged in the system log. Setting is restricted to user root.

```
data,err = systemMgmt.mem(addr[,value],width)
```
Used to read/write addresses through /dev/mem. Restricted to user root. The data read [back] is returned on success; otherwise nil and the error code.

```
0 |nil, err | notifyEnum, text, addText, acceptCode,cancelCode =
systemMgmt.notifyInfo([text,addText,acceptCode[,cancelCode]]|[false])
```
Get/Set-interface for "generic" notify. This can be used to display a notify message in the GUI when the printer goes offline. acceptCode and cancelCode are integer numbers from -1-32767. -1 is used to hide the accept/cancel buttons in the GUI. Nb! Both acceptCode and cancelCode must not be -1.

```
status[,error] = systemMgmt.setAppInfo({name="name",version="version"})
```
Sets the name and version of the running application

```
err = systemMgmt.setCertificate(path,type[,remove])
```
Internal function to set a certificate file of type `type` (one of: `systemMgmt.crt.https`, `systemMgmt.crt.rootCA`, `systemMgmt.crt.client`, `systemMgmt.crt.privateKey`, `systemMgmt.crt.pac`).
If `remove` is true, the file will be removed after installation.

```
err = systemMgmt.setHexDumpPrintPath(path)
```
Internal function to request the file @ path to be parsed/printed.

```
err = systemMgmt.setNotifyComplete(true|false)
```
Internal function to set "Complete"-state true or false.

```
n|nil,error = systemMgmt.setSoftkeys(tSoftkeys)
The softkeys can be set from aep with this function. It returns the number of
softkeys set. See systemMgmt.getSoftkeys()
Example:
do
  local tSoftkeys = systemMgmt.getSoftkeys()
  local online=tSoftkeys[tSoftkeys.ONLINE]
  online[online.vkey_LEFT].code=1138
  online [online.vkey_LEFT].name="Magic"
  online [online.vkey_LEFT].nx=true
  online [online.vkey_LEFT].confirm=true
  assert(systemMgmt.setSoftkeys(tSoftkeys)==#tSoftkeys*#online)
end
```

```
ESUCCESS|nil,error,errstr = systemMgmt.setStatus(tStatus)
```
The interface from aep to set the system status. See systemMgmt.getStatus() for a hint on parameters.
To set bit x and clear bit y in notifyStates, one can use `systemMgmt.setStatus({setNotifyOn=x, setNotifyOff=y})`
To set bit(s) and clear bit(s) in runStates, one can use
```
systemMgmt.setStatus({setRunStateOn=x, setRunStateOff=y})
```

```
ret = systemMgmt.testPrint(type)
```
Issue a single test print job, where type is a value from the testPrintEnum .
Example (print a factory test print label):
```
testPrint = systemMgmt.getGuiEnums().testPrintEnum
type = testPrint.FACTORY
systemMgmt.testPrint(type)
```

```
eventcode = systemMgmt.trigger(event[,eventdata])
eventcode = systemMgmt.trigger(event,eventdata,true)
```
Emits a score event with eventdata. `event` can be either a number or a string as defined in systemMgmt.getGuiEnums().messageTypeEnum. If eventdata is omitted, the event will be sent with eventdata -1. The eventdata range is 0x80000000..0x7FFFFFFF when interpreted as signed, and 0x00000000..0xFFFFFFFF when interpreted as positive. If a negative eventdata value is to be sent, use the 3:argument prototype.

```
set_url = systemMgmt.url([url])
```
Updates the url of the browsers listening on websockets and event gui.guiTypeUrl. This must only be used when the printer is offline. The `url` parameter is appended after the top node of the js-app. If called without arguments, it returns the last set url.

```
errno.ESUCCESS = systemMgmt.url(false)
```
Triggers collecting current url in local browser.

```
current_url = systemMgmt.url(true)
```
Returns the [collected] current url in local browser.

```
err|region = systemMgmt.wlanRegion([wlanRegion])
```
Get/Set the WLAN-region. A set may clear the warp snapshot.

19

IO Extensions

This is not supported in TH2. The IO Extensions provides methods to access system methods to perform Linux scp, ssh and wget command operations. This document does not describe the Linux commands in detail; for that use internet sources. The scp and ssh commands are used to perform encrypted data transmission for secure copy and secure shell. The wget command can be used to download files from ftp-servers and to access webservers using the "http://" and "https://"-protocols.

The IOX module must be loaded with require to get started.

```
local iox=require("autoload.iox")
```

This will provide these functions: `wget, scp, ssh, openssl.dgst, openssl.enc, jpg2png, rotate, bmp2png, video_player, install_splash, ntpf, gzip, gunzip.`

wget() - to communicate with web servers [and ftp servers]

```
fh=iox.wget(url[,args[,dryrun]])
```

Used to run the Linux wget command.
`url` the parameter is required and must contain the protocol to use.
`args` more advanced invocations can pass args as `{"key=value","key","key=value"}`. Not all parameters are accepted that are supported by `iox.wget()` The accepted parameters are:

```
{
  ["--continue"]=true,
  ["--header"]="^('[^%c\\']+')$",
  ["--inet4-only"]=true,
  ["--inet6-only"]=true,
  ["--keep-session-cookies"]=true,
  ["--load-cookies /tmp/cookies.txt"]=true,
  ["--no-cache"]=true,
  ["--no-check-certificate"]=true,
  ["--no-cookies"]=true,
  ["--no-proxy"]=true,
  ["--output-document"]=<PATH>,
  ["--password"]="^('[^%c\\']+')$",
  ["--post-data"]="^('[^%c\\']+')$",
  ["--post-file"]=<PATH>,
  ["--prefer-family"]="^(%w+)$",
  ["--save-cookies /tmp/cookies.txt"]=true,
  ["--save-headers"]=true,
  ["--timeout"]="^(%d+)$",
  ["--tries"]="^(%d+)$",
  ["--trigger"]=1,
  ["--user"]="^('[^%c\\']+')$",
```

```
  ["--user-agent"]="^('[^%c\\']+')$",
}
```

For CT4-LX and FX3-LX the following additional arguments are also available:

```
{
  ["--method"]="GET|POST|PUT|DELETE|PATCH|CONNECT|HEAD|OPTIONS|TRACE",
  ["--body-file"]=<PATH>,
  ["--body-data"]="^('[^%c\\']+')$"
}
```

The default for args is `{"--no-check-certificate"}`.
`dryrun` The parameter is used for trying out what the wget invokation would be; it's for development/debug use only.
The `"--trigger=<number>"` + `"--output-document=<path>"`, and an event listener (`require("autoload.evt").aep.wget` event), can be used with the CLNX event mechanism to trigger an event when the request finishes.

Example:

```
print(iox.wget("http://www.google.com",nil,true))
/usr/bin/wget -q -O - --no-check-certificate http://www.google.com
fp=iox.wget("http://www.google.com",{"--prefer-family=IPv4"})
str=fp:read('*a')
fp:close()
print("the response is:"..#str.." bytes long")
```

scp() - to copy files with SSL-encryption between two computers
ssh() - to execute command on remote computer

```
err|fh=iox.scp(rsa_file,src[,dst[,dryrun]])
fh=iox.scp(rsa_file,host,command[,dryrun])
```

These commands invoke the Linux commands scp and ssh.
`rsa_file` a private key file with a corresponding public key to allow the printer to connect to the server.
`src` is a local or remote file path
`dst` is nil or "/dev/stdout" to retrieve the data via a file handle or a valid path to write it to disk.
`host` is the remote host where the command is to be executed
`command` is the command to execute
`dryrun` is true to just compute and display the command arguments

Examples:

```
fp=iox.scp("/ffs/apps/sa/rsa_key","jdoe@192.168.1.1:./file.txt")
print("contents of 'file.txt' is:")
while fp:read(0) do print(fp:read('*l')) end fp:close()
fp=iox.ssh("/ffs/apps/sa/rsa_key","jdoe@192.168.1.1","date")
print("the date is:")
while fp:read(0) do print(fp:read('*l')) end fp:close()
```

The output could be somewhat like this:

```
contents of 'file.txt' is:
file, what else?
the date is:
Fri Dec  4 14:45:54 CET 2015
```

openssl.dgst() - run the Linux command openssl dgst

```
{
    ["-c"]=true,
    ["-r"]=true,
    ["-hex"]=true,
    ["-binary"]=true,
    ["help"]=true,
    ["-hmac"]="",
    ["-non-fips-allow"]=true,
    ["-mac"]="",
    ["-macopt"]="",
    ["-md4"]=true,
    ["-md5"]=true,
    ["-mdc2"]=true,
    ["-ripemd160"]=true,
    ["-sha"]=true,
    ["-sha1"]=true,
    ["-sha224"]=true,
    ["-sha256"]=true,
    ["-sha384"]=true,
    ["-sha512"]=true,
    ["-whirlpool"]=true,
  }
{type=<type>,v=<v>,raw=<raw>}=iox.openssl.dgst({parameters})
```

This command executes the Linux commands openssl dgst with the supplied parameters.

Example:

```
s=io.open("/ffs/apps/sa/main.lua")
s=s:read('*a')
print(json.encode(iox.openssl.dgst({file="/ffs/apps/sa/main.lua","-hmac
secretkey"})))
{"raw":"HMAC-SHA256(/ffs/apps/sa/main.lua)
c3922742c71db4eaba87d9a1aaffc2c30dd9531454680e086509b96a089fc46f",
"type":"HMAC-SHA256",
"v":"c3922742c71db4eaba87d9a1aaffc2c30dd9531454680e086509b96a089fc46f"} 0
print(json.encode(iox.openssl.dgst({data=s,"-hmac secretkey"}))
{"raw":"HMAC-SHA256(/tmp/lua_lQkXXP))
c3922742c71db4eaba87d9a1aaffc2c30dd9531454680e086509b96a089fc46f",
"type":"HMAC-SHA256",
"v":"c3922742c71db4eaba87d9a1aaffc2c30dd9531454680e086509b96a089fc46f"} 0
print((iox.openssl.dgst({data=s,"-hmac secretkey","-binary"})).raw:byte(1,-1))
195 146 39  66  199 29  180 234 186 135
217 161 170 255 194 195 13  217 83  20
84  104 14  8   101 9   185 106 8   159
196 111
```

openssl.enc() - run the Linux command openssl enc (limited to cipher AES-256-CBC and base64).

```
{
    ["e"]=true, -- encrypt
    ["d"]=true, -- decrypt.
    ["help"]=true, -- show help (or no parameters).
    ["in"]="<input file>", -- input source file for encrypt/decrypt.
    ["out"]="<output file>", -- output source file for encrypt/decrypt.
    ["data"]="<data>", -- input source data for encrypt/decrypt.
    ["pass"]="<pass phrase source>", -- See,
https://www.openssl.org/docs/manmaster/man1/openssl.html (Pass Phrase)
  }
res =iox.openssl.enc({parameters})
```

This command executes the Linux commands openssl enc with the supplied parameters.

Example:

A. Encrypt /tmp/file.txt to /tmp/file.txt.ssl.
```
res = iox.openssl.enc({e=true, pass="pass:mypassword",
["in"]="/tmp/file.txt", out="/tmp/file.txt.ssl"})
```
B. Decrypt /tmp/file.txt.ssl to /tmp/file.txt.ssl.txt.
```
res = iox.openssl.enc({d=true, pass="pass:mypassword",
["in"]="/tmp/file.txt.ssl", out="/tmp/file.txt.ssl.txt"})
```
C. Encrypt /tmp/file.txt to return result.
```
res = iox.openssl.enc({e=true, pass="file:/tmp/password.key",
["in"]="/tmp/file.txt"})
```
D. Decrypt data to /tmp/file.txt.ssl.txt.
```
res = iox.openssl.enc({d=true, pass="file:/tmp/password.key", data=res,
out="/tmp/file.txt.ssl.txt"})
```

openssl.enc is used preferable together with pcall to catch errors.

newfile=jpg2png(jpgimagefile) - convert jpg image to png image
newfile=rotate(jpgimagefile|pngimagefile,rotation) - convert image (jpg|png) and rotate it

These commands are implemented using Python Imaging Library. They create a file in /tmp each time they are invoked, and the created file name is returned. It is your responsibility to delete the created files when they are not used anymore.

pngPath = iox.bmp2png(bmpPath, removeOriginal)
Convert a bmp image to a png image. If removeOriginal is true, the input file (bmpPath) will be removed after conversion. Returns the path of the newly created png file.

errno = iox.video_player(url, options)
Play a video file, specified by the string *url*, on the printer (FX3). Set options, specified by the table *options*. The following options (strings) are supported:
"-c": close on done (end of media).
"-l": landscape orientation.

"-p": portrait orientation.
"-r": repeat/loop video.
"-s": start in the stopped playback state, i.e. auto-play off.

Returns 0 on success (this does not necessarily mean that the video is playing successfully, see the system log in case of an error that is not showed on the display).
Only one instance of a video player can run at a time.

```
errno = iox.install_splash(path, startup)
```

Only available/implemented on FX3-LX and CT4-LX. Installs a startup or shutdown image on the printer. File at path is not deleted. Requires restart after installation to show the images.

`path` = path to correctly sized png image. `path = ""` removes corresponding image.
`startup` = true for startup image.
`startup` = false for shutdown image.
`errno` = Error number. Returns errno.ESUCCESS (0) when successful.

```
l = iox.ntpf()
```

This command runs ntp in burst mode to synchronize the printer time with an NTP-server. It is intended to be used in PW2NX only. It requires a working network connection; if it cannot be detected, nil is returned. Otherwise is a string containing information about how much the time was adjusted.

```
iox.timepatch(callbacks)
```

This command is used to enable real time tracking in PW2NX. It is used by the Standard Application from 3.2.0-r1.

```
<error code> [, <FILE>] = iox.gzip([<source_file> ,] <target_file>)
```
Compress a file (or other data) to `<target_file>`. `<target_file>` is created/overwritten.

If only `<target_file>` is supplied, a file handle is returned that the source data shall be written to (`FILE:write(...)`). When all data is written, the FILE handle shall be closed (`FILE:close()`)

If both source and target files are supplied, `<source_file>` is used as input.
`<source_file>` is not modified nor removed.

```
<error code> [, <FILE>] = iox.gunzip(<source_file> [, <target_file>])
```
Decompress a (gzip compressed) file to `<target_file>` or FILE stream.
`<source_file>` is not modified nor removed.

If only `<source_file>` is given, a file handle is returned that the data can be read from (`FILE:read()`).

If both source and target files are supplied, `<target_file>` is created/overwritten and will contain the decompressed data.

20

Logger

The logger is used for development to output messages to the volatile system log. This is not supported on TH2. WebConfig will show the collected logs in Support Info. The interface:

```
logger=require("autoload.logger")
logger.ident(name)
logger.log(msg[,level]) or logger(msg[,level])
logger.loglevels(levels)
logger[level]
logger.ranges()
```

```
logger=require("autoload.logger")
```

Loads the logger module

```
logger.ident(ident)
```

Sets the identity for the current process (see man 3 openlog). SA uses `"aep-SA"` as identity.

```
logger(msg [,level])
logger.log(msg[,level])
```

Appends the string msg to the system log if the loglevel `level` is enabled. If `level` is omitted it defaults to info. Typical usage:

```
logger("serious error",logger.E)
logger("warning message",logger.W)
logger("This is just informational")
```

```
levels=logger.loglevels()
logger.loglevels(level1,level2,...)
logger.loglevels({level1,level2,...})
```

Returns the enabled loglevels as a table or enables the levels provided. The default loglevel logs errors. It's possible to programmatically enable other levels by the below pattern:

```
do
  local t=json.decode(configTbl.aep.storage)
  t=type(t)=="table" and t or {}
  t.aep=type(t.aep)=="table" or {}
  t.aep.loglevels={"e","i","w"}
  configTbl.aep.storage=t
end
```

```
logger[level]
```

Levels can be referenced as

```
logger.E,logger.e,logger.err,logger.ERR,logger.ERROR,logger.error
logger.W,logger.w,logger.warn,logger.WARN,logger.WARNING,logger.warning
```

```
logger.I,logger.i,logger.info,logger.INFO

ranges=logger.ranges()
```

Provides some information about log levels

```
print(json.encode(logger.ranges(),nil,true))
{
 "loglevels":{
  "WARN":"-- warning",
  "warning":"-- warning",
  "w":"-- warning",
  "warn":"-- warning",
  "e":"-- error",
  "E":"-- error",
  "W":"-- warning",
  "WARNING":"-- warning",
  "INFO":"-- info",
  "i":"-- info",
  "info":"-- info",
  "I":"-- info",
  "ERROR":"-- error",
  "err":"-- error",
  "error":"-- error",
  "ERR":"-- error"
 }
}
```

21

Pack

The pack library have a few functions to convert low-level data types into other formats in CLNX. Examples:

```
> return (pack.toBytes(1.0):gsub(".",function(x) return
string.format("%2.2x",x:byte()) end))

000000000000f03f

> return pack.toNumber(pack.toBytes(1.0)..pack.toBytes(2.0))

1.0        2.0

> return pack.toInt32(string.char(0x80,0,0,0, 0,1,0,0))

128        256
```

The methods:

toNumber()
```
n1,n2,...=pack.toNumber(str[,n])
```

The method converts bytes in str into native doubles. If n is given, that number of lua numbers are returned. This is supported also in TH2.

toInt32()
```
n1,n2,...=pack.toInt32(str[,n])
```

The method converts bytes in str into native int32_t converted to lua "number". If n is given, that number of int32_t are returned (convert to lua type number).

toBytes()
```
str=pack.toBytes(number[,bigEndian])
```

The method converts the lua number into its byte representation. If bigEndian is given, the byte stream is returned in big endian byte order. If bigEndian is omitted or false, the little endian byte order is used.

22

pkgObject

The pkgObject is available internally for building pkg files. It is documented here for internal purposes.

```
obj,err = pkgObject.new([<pkg file path>])
```
Creates a new pkgObject or from a pkg file.

```
success (bool) [, errno] = obj:addFile(<local_path>, <target_path>
                           [, <user> (default 'user'),
                              <mode> (octal string or direct value) (default
"0644")]
                           [, <compress> (default true)])
```
Adds a file to be added to the pkg object. EPERM if at least one entry is of type DIG.

```
success (bool) [, errno] = obj:addLuaFile(<local_path> [, <compress> (default
true) ])
```
Adds a lua entity to be added to the pkg object.

```
success (bool) [, errno] = obj:addLua(<lua_string> [, <compress> (default true)
])
```
Adds a lua string to be added to the pkg object.

```
success (bool) | {success = true|false, errno = <errno>} = obj:add([array])
```

Add multiple items from the array. If any item cannot be accessed correctly (missing file, wrong permissions), the return value will be an array of return values, one entry for each item in [array]. The items are described as:
`{type='File|FILE|Lua|LuaFile|LUAC',path=filepath,compress=true|false|nil,target=target_path,user=user,mode=mode,code="lua code"}`
where the applicable attributes can be mapped to the other add methods.

`success (bool) [, errno] = obj:write(<pkg_path>)`
Builds the pkg file from the added items, and writes to disk. Identical entities share the same pkg blob. EEXIST if file exists and is a source in package.

`[array] = obj:list()`
Returns an array of items added to the pkg object.
Array type can be one of the following;
`BOOT,KERN,FILE,LUAC,LAN,WLAN,RPM,DIG,PLT,FS1,FS2,PUB,TGT,RAM,RLE,RGE,FX3`

```
{
type = "LUAC",
code = <code> | path = <local path>
[, compress = true|false (default true)]
}

{
type = "FILE",
path = <local path>,
target = <printer path>
[, user = <name> (default 'user')]
[, mode = '644' 420 (octal string or direct value)]
[, compress = true|false (default true)]
}

{
type =
"BOOT"|"KERN"|"LAN"|"WLAN"|"RPM"|"DIG"|"PLT"|"FS1"|"FS2"|"PUB"|"TGT"|"RAM"|"RLE"
|"RGE"|"FX3"
path = <local package path>
[, compress = true|false (default true)]
}
```

`success (bool) [, errno] = obj:insert(idx, <entry>)`
Insert entry at index idx [1..#list+1].
Only FILE and LUAC entries supported.EPERM if at least one object entry is of type DIG. Entry defined as reported by `list()`.

`success (bool) [, errno] = obj:remove(idx)`
Remove entry at index idx [1..#list]. EPERM if at least one object entry is of type DIG.

`success (bool) [, errno] = obj:replace(idx, <entry>)`
remove entry at index idx [1..#list].

Only FILE and LUAC enties supported. EPERM if at least one object entry is of type DIG. Entry defined as reported by `list()`.

23

lsrender

When the lsrender module is loaded, aep can utilize more rendering resources available in the common SATO render module, used by SBPL and other emulators.

When loading the lsrender module, the rendering API is extended with additional barcodes, and bitmap fonts. When everything is enabled, human readables in barcodes are in general slightly different compared to when disabled, so each renderer can be individually controlled.

The barcodes made available by using lsrender:

barcodeObject.newAztec()
barcodeObject.newCode93()
barcodeObject.newCustomer()
barcodeObject.newGs1DatabarE()
barcodeObject.newGs1Datamatrix()
barcodeObject.newInd2of5()
barcodeObject.newMSI()
barcodeObject.newMicroPdf417()
barcodeObject.newNEC2of5()
barcodeObject.newPOSTNET()
barcodeObject.newQrcodeModel1()
barcodeObject.newSQrcode()
barcodeObject.newUSPS()

23.1 Enabling lsrender

In order to use lsrender, it needs to be loaded:

```
require("lsrender")
```

This loads lsrender, and enables all renderers.

23.2 Controlling renderers

lsrender can be enabled or disabled on the fly. When a renderObject is added to a labelObject, it is locked, and the lsrender state can be switched again.

Disable all lsrender renderers:

```
require("lsrender").ctrl(false)
```

Enable all lsrender renderers:

```
require("lsrender").ctrl(true)
```

Inspect and visualize renderers:

```
local list=require("lsrender").ctrl()
for k,v in pairs(list) do print (k,(json.encode(v))) end

bcMaxiCode_srender     {"on":true}
bcInt2of5_srender      {"on":true}
obj_strFontset {"on":true}
bcEan_srender  {"on":true}
bcCodabar_srender      {"on":true}
bcBookland_srender     {"on":true}
bc128C_srender {"on":true}
bcDataMatrix_srender   {"on":true}
bcQRCode_srender       {"on":true}
obj_str2HumanReadableFontID    {"on":true}
obj_strTypeface{"on":true}
pdf417_srender {"on":true}
srender_init   {"on":true}
bcCode39_srender       {"on":true}
edit_ank_srender       {"on":true}
edit_knj_srender       {"on":true}
bcDatabar_srender      {"on":true}
bcCode128_srender      {"on":true}
bcEanC_srender {"on":true}
```

Individual items can be disabled/enabled, e.g.

```
list.bcEanC_srender.on=false
require("lsrender").ctrl(list)
```

23.3 Bitmap fonts

lsrender also provides access to additional bitmap fonts. They can be enumerated with bmfonts, and used with the textBMObject API.

```
local fonts=require("lsrender").bmfonts()
-- each table entry has {id=id,name=name}
-- the name can be used in the bitmap font API
local bm=textBMObject.new("MRM8")
```

24

Non supported

Dynamic loading of linkable libraries loaded with `require` is not supported on TH2. Some functions in the os library are not supported:

`getenv` - Does not return any values in the printer on TH2 (but works in shared code and as a means to distinguish between emulated environment (Psim/TH Works) and firmware.

`execute` - not supported on TH2

`rename` - not supported on TH2

`exit` - not supported on TH2

`setlocale` - not supported

25

Document

25.1 References

[1] TH2 System Specification, STB00029.
[2] TH2 Operation specification, STB00028.
[3] TH2 Command Channel, STB00023.
[4] TH2 Error Handling, STB00095.
[5] LuaSocket, http://www.tecgraf.puc-rio.br/~diego/professional/luasocket
[6] lua-websockets, http://lipp.github.io/lua-websockets
[7] LuaSec, https://github.com/brunoos/luasec/wiki
[8] lua-ev, https://github.com/brimworks/lua-ev
[9] LuaSQL, https://keplerproject.github.io/luasql/
[10] SQLite, http://sqlite.org/

25.2 Revision history

Revision	Name	Comment	Date
PA1	Lars-Åke Berg	Created	2007-06-20
PC1	Per Andersson	Version PC1 and onwards describe firmware later than version 40.00.02.00.	2012-05-16
PC2	Per Andersson	Added method “angles” for object circle and ellipse.	2012-05-22
PC3	Lars Persson	Added bignum support.	2012-05-25
PC4	Lars Persson	Added neg() and cmp() for bignum support. Changed “was is” to “is” in chapter 7.3.40.2, angles().	2012-05-29
PC5	Martin Dahlberg	Sync with PB47 - add model to system.info(). Clarified limitations on Maxicode & PDF417	2012-06-20
PC6	Lars-Åke Berg	Added fs.mkdir create parent directory option.	2012-07-04
PC7	Per Andersson	Corrected typos found by QA. Clairified behavior when barcodes are positioned partly outside printable area (5.3.1.2). Added note on pen mode REPLACE and barcodes (5.3.1.5). Clarified max data limitations for barcode MaxiCode (5.3.2.9).	2012-07-05
PC8	Martin Dahlberg	Added section to text true type on supported encodings	2012-07-06
PC9	Per Andersson	Added method “clone” for all render objects (text, text box, barcode, line, box, image, circle, and ellipse).	2012-08-31

		Removed “pen” from description of text box.	
PC10	Per Andersson	Removed functions wlan.toCard(), wlan.fromCard(), wlan.set(), and wlan.get(). Changed name of function from wlan.info() to wlan.getInfo().	2012-12-13
PC11	Victor Hagsand	Added new functionality in setTime and Buzzer functions for a new printer	2013-10-17
PC12	Martin Dahlberg	Added engine.skipMode and textBoxObject.gc, textBoxObject.enableCache -descriptions	2013-11-12
PC13	Martin Dahlberg	Added job.stats -description. Updated system.sound-description. Updated system.setTime Added System Management chapter	2013-11-15
PC14	Martin Dahlberg	Updated System Management chapter. Updated system.info(). Grayed out some sections for deletion.	2014-04-23
PC15	Per Andersson	A parameter is added in function system.power() to decide whether a cached or the current coin battery state should be returned.	2014-06-12
PC16	Per Andersson	Added "maxLength" in description of "engine.canvasInfo()".	2014-08-15
PC17	Martin Dahlberg	Added information on system.info, systemMgmt.getMDL and system.upgrade	2014-08-27
PC18	Fredrik Johansson	Updated parameters for config.write.	2014-09-03
PC19	Fredrik Johansson	Added information on systemMgmt.testPrint	2014-09-09
PC20	Martin Dahlberg	Added information in systemMgmt, system.upgrade and Device	2014-10-10
PC21	Victor Hagsand	Added append and grab to device.open	2014-10-15
PC22	Magnus Wibeck	systemMgmt.setCertificate's third argument. systemMgmt.setStatus{setNotify*=x}	2014-11-12
PC23	Martin Dahlberg	Added description of config.reset()	2015-02-16
PC24	Fredrik Johansson	Added description of i18nObject decode method.	
PC25	Lars-Åke Berg	Added system.ahd and update of device.open.	2015-04-16
PC26	Lars-Åke Berg	Added rx, tx and connects to system.ahd.	2015-04-23
PC27	Martin Dahlberg	Updated fs, engine, job, system with information about TH2 and CLNX-products.	2015-07-07
PC28	Lars Persson	Added the ZIP method close().	2015-08-17
PC29	David Holmin	Added documentation for CLxNX GUI	2015-08-28
PC30	David Holmin	Updated CLxNX GUI with new information on the select type.	2015-09-01

PC31	Martin Dahlberg	Updated CLxNX GUI and added CLxNX Event System and a CLxNX AEP applications sections	2015-09-09
PC32	Mats Hedberg	Changed front page	2015-09-17
PC33	David Holmin	Updated the section about "sendkeys", under CLxNX GUI	2015-09-18
PC34	Per Andersson	Added description of function job.runSbplFromAep().	2015-09-25
PC35	Victor Hagsand	Added section about extKbd	2015-09-30
PC36	Martin Dahlberg	Reviewed sections in sdb, extKbd, job, system, systemMgmt	2015-10-23
PC37	Lars-Åke Berg	tlcounterOffset added to systemMgmt.get|setStatus	2015-12-08
PC38	Martin Dahlberg	Added information about HTTPS-support, iox, json, config.write, sdb and sdbObject.	2015-12-21
PC39	Per Andersson	Added canvas:suppressOffsets().	2016-01-18
PC40	Martin Dahlberg	Added descriptions for sdb.offset(), sdb.wrap() and system.newEvents() Removed revision history for PA2-PB99	2016-01-21
PC41	David Holmin Martin Dahlberg	Updated info on f1 and f2 under CLxNX GUI Minor spelling corrections. Reformatted CLxNX events	2016-01-22
PC42	Martin Dahlberg	Added system views and introduction to Lua	2016-02-23
PC43	Martin Dahlberg	Corrections and additions in system views and Lua introduction	2016-02-25
PC44	Martin Dahlberg	Additions in system views (the states AEP, ONLINE, OFFLINE, ERROR, SETTINGS) Additions in gui	2016-03-01
PC45	Lars-Åke Berg	Addition of <textTTObject>:face() method and update of <textTTObject>:info().	2016-04-08
PC46	Victor Hagsand	Addition of <Object>:ranges() method. Made style of object default lists consistent. Added missing defaults for text object.	2016-04-13
PC47	Martin Dahlberg	Updated the CLxNX gui descriptions	2016-04-15
PC48	Victor Hagsand	Fixed some errors in ranges descriptions.	2016-04-15
PC49	Victor Hagsand	Added “Not available on TH2.” To all ranges descriptions.	2016-04-15
PC50	Victor Hagsand	Added example to ranges and minor cleanup.	2016-04-22
PC51	Martin Dahlberg	Modifications in device section	2016-05-11
PC52	Lars-Åke Berg	<textTTObject>.new, :font and :face API update. Added Font resources (system.resourceInit(), system.resource and textTTObject extension).	2016-05-12
PC53	Martin Dahlberg	Updated the CL4NX events object description and added the Logger chapter	2016-05-12
PC54	Victor Hagsand	Shortened ranges descriptions.	2016-05-17

PC55	Lars-Åke Berg	Autohunter port 1024 & 9100 dependent on configTbl.network.lan.port_queue.	2016-05-19
PC56	Martin Dahlberg	Described pack functions	2016-05-25
PC57	Lars Persson	Support 32-bit PNG images by stripping the alpha channel.	2016-06-17
PC58	Magnus Wibeck	Table editing - sdb.addCol(), sdb.deleteCol(), sdb.changeTable().	2016-09-09
PC59	Martin Dahlberg	Updates on json.decode, events:hook	2016-09-23
PC60	Magnus Wibeck	System.linkStandardApp()	2016-11-15
PC61	Lars Persson	Updated engine.mstat()	2016-11-29
PC62	Per Andersson	Added new textbox method "wrapChars" and updated definition of "hyphen".	2016-12-21
PC63	Per Andersson	Added the new render object "grid".	2017-02-07
PC64	Martin Dahlberg	string.length => string.len, Lua 5.1.4, http-url to Expat, iox openssl.dgst and rotate/jpg2png, gui, toBmp, fs.sync, fs.realpath	2017-02-09
PC65	Martin Dahlberg	Updated to Lua 5.1.5, &reply=t examples added to CLNX GUI	2017-03-09
PC66	Per Andersson	Added method "radius" in description of the Box object.	2017-03-09
PC67	Martin Dahlberg	Added events:poll_add, events:poll and described new option --trigger in iox.wget	2017-03-17
PC68	Martin Dahlberg	Described new argument job.poll(true), added the internal documentation of pkgObject	2017-03-30
PC69	Per Andersson	Corrected typos: bit.blshift -> bit.lshift and bit.brshift -> bit.rshift.	2017-04-10
PC70	Lars Persson	Added additional argument to sdb.XLSXToXML() and one more possible error response.	2017-04-25
PC71	Lars Persson	Updated and added additional argument to sdb.XLSXToXML().	2017-04-26
PC72	Per Andersson	Added logData.	2017-05-02
PC73	Per Andersson	Added note about settings used when running job.runSbplFromAep. Added field 'formatter' in description of logData.	2017-05-03
PC74	Per Andersson	Removed logData. **This revision is the best match for 1.9.0 - 1.9.6.**	2017-05-08
PD1	Lars Persson	Updated BIT support with Lua BitOp.	2017-05-16
PD2	Lars Persson	Added lua-websockets, LuaSec, and lua-ev.	2017-06-09
PD3	Martin Dahlberg	Updated systemMgmt.trigger() with a 3:rd argument	2017-06-16
PD4	Lars-Åke Berg	Embolden and Oblique textTTObject methods.	2017-08-29
PD5	Per Andersson	Added description of LuaSQL and SQLite.	2017-09-19

PD6	Fredrik Johansson	Added description of iox.bmp2png	2017-10-12
PD7	Lars Persson	Added description of iox.video_player.	2017-10-13
PD8	Lars Persson	Added the options argument to iox.video_player.	2017-11-09
PD9	Magnus Wibeck	System.password for CLNX, PWNX, FX3.	2017-11-14
PD10	Magnus Wibeck	iox.install_splash() for CLNX, PWNX, FX3.	2017-11-30
PD11	Martin Dahlberg	iox.ntpf primarily for PW2NX	2018-01-30
PD12	Martin Dahlberg	Added items in sdb and Excel import	2018-03-16
PD13	Martin Dahlberg	Added iox.timepatch	2018-04-12
PD14	Lars-Åke Berg	Added iox.openssl.enc	2018-04-19
PD15	Per Andersson	Fixed error in parameter order for systemMgmt.mem().	2018-06-26
PD16	Magnus Wibeck	iox.install_splash() not yet implemented for CLNX, PWNX **This revision is the best match for: 1.10.0-r1, 5.0.3-r1, 3.2.2-r2**	2018-08-21
PE1	Lars-Åke Berg	<textTTObject>:shaper([true|false]) added.	2018-09-28
PE2	Lars-Åke Berg	<textTTObject>:shaper() additionaly return cluster table if shaper is enabled.	2018-10-11
PE3	Lars Persson	Added the "-p" option to iox.video_player.	2018-11-19
PE4	Lars-Åke Berg	Updated <textTTObject> shaper description.	2019-01-30
PE5	Lars-Åke Berg	Updated pkgObject.new method. Added methods pkgObject: insert, remove and replace.	2019-03-06
PE6	Magnus Wibeck	iox.wget accepts --method, --body-data, and --body-file. CT4-LX and FX3-LX only.	2019-03-15
PE7	Martin Dahlberg	Added descriptions to device.	2019-04-16
PE8	Magnus Wibeck	systemMgmt.getStatus().runState	2019-05-07
PE9	Ahmed Nuur	Updated pkgObject.new method (FILE, LUAC).	2019-07-11
PE10	Ahmed Nuur	Added dir method to imageObject.	2019-09-06
PE11	Ahmed Nuur	Added default value to imageobject dir.	2019-09-09
PE12	Magnus Wibeck	Added iox.gzip/iox.gunzip (expected in 1.12.0, 3.4.0, 5.2.0, 6.3.0).	2019-09-10
PE13	Fredrik Johansson	Added USB path information for various models.	2019-09-30
PE14	Lars-Åke Berg	Added LuaSec https proxy support.	2019-10-09
PE15	Lars-Åke Berg	Update of LuaSec https proxy description.	2019-10-11
PE16	Lars-Åke Berg	Proxy authorization to LuaSec. Proxy and authorization to luawebsocket.	2019-10-25
PE17	Martin Dahlberg	Updates related to lsrender in barcode and bitmap descriptions.	2019-11-14

Extensive contact information of worldwide SATO operations can be found on the Internet at **www.satoworldwide.com**

SATO