

Getting started

luatex is a typesetter; texlua and luatex --luaonly are lua interpreters. In lua interpreter mode, the lua tables tex, token, node, and pdf are unavailable.

\directlua expands immediately, \atlua is processed during \shipout

\luatexversion: 78
 \luatexrevision: 3
 \luatexdatestamp: 2014031011

Attributes

There are 65536 attribute registers, each of which obeys grouping and can have any integer value other than the most negative number (which indicates the value is unset).

Category tables

There are 65536 category code tables, each of which can hold the full range of Unicode. Category table manipulation:

\initcatcodetable initialize an 'initex' table in
 \savecatcodetable save the current category codes to table globally
 \catcodetable switch to predefined table

File names

\input, \openin, and \font accept braced file names to allow embedded spaces.

Images and forms

\pdfrefximage and \pdfrefxform accept optional dimension parameters in the same format as \pdfximage.

Preloaded lua modules

slunicode <http://luaforge.net/projects/slunicode/>
 luazip <http://www.keplerproject.org/luazip/>
 luafilesystem <http://www.keplerproject.org/luafs/>
 lpeg <http://www.inf.puc-rio.br/~roberto/lpeg.html>
 lzlib <http://luaforge.net/projects/lzlib/>
 md5 <http://www.inf.puc-rio.br/~roberto/md5/md5-5/md5.html>
 luasocket <http://www.tecgraf.puc-rio.br/~diego/professional/luasocket/>

String extensions

table = string.explode(string, [string])
 Break a string into pieces. The optional argument is a character possibly followed by a plus sign (default "+")
 number = string.utfvalues(string)
 Iterator that returns a value representing a single UTF-8 token.
 string = string.utfcharacters(string)
 Iterator that returns a string representing a single UTF-8 token.
 string = string.characters(string)
 Iterator that returns a string representing a single 8-byte token.
 string, string = string.characterpairs(string)
 Iterator that returns two strings representing two single UTF-8 tokens.
 number = string.bytes(string)
 Iterator that returns a value representing a single 8-byte token.
 number, number = string.bytepairs(string)

Iterator that returns two values representing two single 8-byte tokens.

Operating system extensions

os.exec(table)
 Run an external command and exit. The table is an array of arguments, with an optional argv[0] in index 0.
 boolean = os.spawn(table)
 Run an external command and return its exit code. The table is an array of arguments, with an optional argv[0] in index 0.
 os.setenv(string, string) Set an environment variable.
 number = os.gettimeofday()
 Get the time as a floating point number (Unix only).
 table = os.times() Return process times.
 string = os.tmpdir()
 Create a temporary directory inside the current directory.
 table = os.uname()
 Return various information strings about the computer.
 string = os.selfdir() Return the directory path of argv[0].

Lua file system extensions

boolean = lfs.isdir(string) Return true if the string is a directory.
 boolean = lfs.isfile(string) Return true if the string is a file.
 string = lfs.shortname(string)
 Return the FAT name of a file (Windows only).
 string = lfs.readlink(string)
 Return the contents of a symlink (Unix only).

Callback table

number, [string] = callback.register(string, function)
 Register a callback. Passing nil removes an existing callback. Returns nil, error on failure.
 table = callback.list() Produce a list of all known callback names.
 function = callback.find(string)
 Returns the function currently associated with a callback, or nil

File discovery callbacks

string = find_read_file(number, string)
 Find a file for \input (0) or \openin (higher integers).
 string = find_write_file(number, string)
 Find a file for writing to the log file (0) or with write (higher integers).
 string = find_font_file(string) Find a font metrics file.
 string = find_output_file(string) Find the output (PDF or DVI) file.
 string = find_format_file(string) Find the format file.
 string = find_vf_file(string) Find a VF file.
 string = find_map_file(string) Find a font map file.
 string = find_enc_file(string) Find a font encoding file.
 string = find_subfont_file(string) Find a subfont definition file.
 string = find_pk_file(string) Find a PK font bitmap file.
 string = find_data_file(string)
 Find an input data file for PDF attachment.
 string = find_opentype_file(string) Find an OpenType font file.
 string = find_truetype_file(string) Find a TrueType font file.
 string = find_type1_file(string) Find a Type1 (PostScript) font file.
 string = find_image_file(string) Find an image file for inclusion.

File reading callbacks

table = open_read_file(string)

Open a file for reading. The returned table should define key functions for "reader" and "close".

`string = reader(table)`
Read a line from a file opened with the `open_read_file` callback. The argument is the return value from `open_read_file`

`close(table)`
Close a file opened with the `open_read_file` callback. The argument is the return value from the `open_read_file`

`boolean, string, number = read_font_file(string)`
Read a TFM metrics file. Return true, the data, and the data length for success, false otherwise

`boolean, string, number = read_vf_file(string)` Read a VF metrics file.

`boolean, string, number = read_map_file(string)` Read a font map file.

`boolean, string, number = read_enc_file(string)`
Read a font encoding file.

`boolean, string, number = read_sfd_file(string)`
Read a subfont definition file.

`boolean, string, number = read_pk_file(string)`
Read a font bitmap PK file.

`boolean, string, number = read_data_file(string)` Read a data file.

`boolean, string, number = read_truetype_file(string)`
Read a TrueType font

`boolean, string, number = read_type1_file(string)` Read a Type1 font

`boolean, string, number = read_opentype_file(string)`
Read an OpenType font

Tokenisation changes callbacks

`string = process_input_buffer(string)`
Modify the encoding of the input buffer.

`string = process_output_buffer(string)`
Modify the encoding of the output buffer.

`table = token_filter()`
Override the tokenization process. Return value is a token or an array of Font definition callback tokens

Node list callbacks

`buildpage_filter(string)`
Process objects as they are added to the main vertical list. The string argument gives some context

`buildpage_filter` context information:

value	explanation
<code>alignment</code>	a (partial) alignment is being added
<code>after_output</code>	an output routine has just finished
<code>box</code>	a typeset box is being added
<code>new_graf</code>	the beginning of a new paragraph
<code>vmode_par</code>	¶par was found in vertical mode
<code>hmode_par</code>	¶par was found in horizontal mode
<code>insert</code>	an insert is added
<code>penalty</code>	a penalty (in vertical mode)
<code>before_display</code>	immediately before a display starts
<code>after_display</code>	a display is finished
<code>end</code>	LUA _{TEX} is terminating (it's all over)

`node = pre_linebreak_filter(node, string)`
Alter a node list before linebreaking takes place. The string argument gives some context

`pre_linebreak_filter` context information:

value	explanation
<code><empty></code>	main vertical list
<code>hbox</code>	¶hbox in horizontal mode
<code>adjusted_hbox</code>	¶hbox in vertical mode
<code>vbox</code>	¶vbox

<code>vtop</code>	¶vtop
<code>align</code>	¶halign or ¶valign
<code>disc</code>	discretionaries
<code>insert</code>	packaging an insert
<code>vcenter</code>	¶vcenter
<code>local_box</code>	¶Vocalleftbox or ¶Vocalrightbox
<code>split_o</code>	top of a ¶split
<code>split_keep</code>	remainder of a ¶split
<code>align_set</code>	alignment cell
<code>fin_row</code>	alignment row

`node = linebreak_filter(node, boolean)`
Override the linebreaking algorithm. The boolean is true if this is a pre-display break.

`node = post_linebreak_filter(node, string)`
Alter a node list after linebreaking has taken place. The string argument gives some context

`node = hpack_filter(node, string, number, string, string)`
Alter a node list before horizontal packing takes place. The first string gives some context, the number is the desired size, the second string is either "exact" or "additional" (modifies the first string), the third string is the desired direction

`node = vpack_filter(node, string, number, string, number, string)`
Alter a node list before vertical packing takes place. The second number is the desired max depth. See `hpack_filter` for the arguments.

`node = pre_output_filter(node, string, number, string, number, string)`
Alter a node list before boxing to ¶outputbox takes place. See `vpack_filter` for the arguments.

`hyphenate(node, node)` Apply hyphenation to a node list

`ligaturing(node, node)` Apply ligaturing to a node list

`kerning(node, node)` Apply kerning to a node list

`node = mlist_to_hlist(node, string, boolean)`
Convert a math node list into a horizontal node list

`metrics = define_font(string, number)`
Define a font from within lua code. The arguments are the user-supplied information, with negative numbers indicating scaled, positive numbers at

Event callbacks

`pre_dump()` Run actions just before format dumping takes place.

`stop_run()` Run actions just before the end of the typesetting run.

`start_run()` Run actions at the start of the typesetting run.

`start_page_number()`
Run actions at the start of typeset page number message reporting.

`stop_page_number()`
Run actions at the end of typeset page number message reporting.

`show_error_hook()` Run action at error reporting time.

`finish_pdf_file()` Run actions just before the PDF closing takes place.

Epdf table

--- All constructors:

`PDFDoc = epdf.open(string)`
Construct a PDFDoc object by opening a PDF document

`Annot = epdf.Annot(XRef, Dict, Catalog, Ref)`
Construct an Annot object

`Annots = epdf.Annots(XRef, Catalog, Object)`
Construct an Annots object

`Array = epdf.Array(XRef)` Construct an Array object

`Dict = epdf.Dict(XRef)` Construct a Dict object

```

Object = epdf.Object()    Construct an Object object
PDFRectangle = epdf.PDFRectangle()
    Construct a PDFRectangle object
--- Annot methods:
boolean = Annot.isOK()    Check if A nnot object is ok.
Object = Annot.getAppearance()    Get A ppearance object
AnnotBorder = Annot.getBorder()    Get A nnotBorder object
boolean = Annot.match(Ref)
    Check if object number and generation matches Ref.
--- AnnotBorderStyle methods:
number = AnnotBorderStyle.getWidth()    Get border width.
--- Annots methods:
integer = Annots.getNumAnnots()    Get number of A nnots objects
Annot = Annots.getAnnot(integer)    Get A nnot object
--- Array methods:
Array.incRef()    Increment reference count to A rray.
Array.decRef()    Decrement reference count to A rray.
integer = Array.getLength()    Get A rray length.
Array.add(Object)    Add Object to A rray.
Object = Array.get(integer)    Get Object from A rray.
Object = Array.getNF(integer)
    Get Object from A rray, not resolving indirection.
string = Array.getString(integer)    Get String from A rray.
--- Catalog methods:
boolean = Catalog.isOK()    Check if Catalog object is ok.
integer = Catalog.getNumPages()    Get total number of pages.
Page = Catalog.getPage(integer)    Get Page.
Ref = Catalog.getPageRef(integer)
    Get the reference to a Page object.
string = Catalog.getBaseURI()    Get base URI, if any.
string = Catalog.readMetadata()
    Get the contents of the Metadata stream.
Object = Catalog.getStructTreeRoot()
    Get the structure tree root object.
integer = Catalog.findPage(integer, integer)
    Get a Page number by object number and generation.
LinkDest = Catalog.findDest(string)    Find a named destination.
Object = Catalog.getDest()    Get destinations object.
integer = Catalog.numEmbeddedFiles()
    Get number of embedded files.
FileSpec = Catalog.embeddedFile(integer)
    Get file spec of embedded file.
integer = Catalog.numJS()    Get number of javascript scripts.
string = Catalog.getJS(integer)    Get javascript script.
Object = Catalog.getOutline()    Get Outline object.
Object = Catalog.getAcroForm()    Get A croForm object
--- EmbFile methods:
string = EmbFile.name()    Get name of embedded file.
string = EmbFile.description()    Get description of embedded file.
integer = EmbFile.size()    Get size of embedded file.
string = EmbFile.modDate()    Get modification date of embedded file.
string = EmbFile.createDate()    Get creation date of embedded file.
string = EmbFile.checksum()    Get checksum of embedded file.
string = EmbFile.mimeType()    Get mime type of embedded file.
Object = EmbFile.streamObject()
    Get stream object of embedded file.
boolean = EmbFile.isOk()    Check if embedded file is ok.
--- Dict methods:
Dict.incRef()    Increment reference count to Dict.
Dict.decRef()    Decrement reference count to Dict.
integer = Dict.getLength()    Get Dict length.
Dict.add(string, Object)    Add Object to Dict.
Dict.set(string, Object)    Set Object in Dict.
Dict.remove(string)    Remove entry from Dict.
boolean = Dict.is(string)    Check if Dict is of given /Type.
Object = Dict.lookup(string)    Look up Dict entry.
Object = Dict.lookupNF(string)

```

```

    Look up Dict entry, not resolving indirection.
integer = Dict.lookupInt(string, string)    TODO
string = Dict.getKey(integer)    Get key from Dict by number.
Object = Dict.getVal(integer)    Get value from Dict by number.
Object = Dict.getValNF(integer)
    Get value from Dict by number, not resolving indirection.
--- Link methods:
boolean = Link.isOK()    Check if Link object is ok.
boolean = Link.inRect(number, number)
    Check if point is inside the link rectangle.
--- LinkDest methods:
boolean = LinkDest.isOK()    Check if LinkDest object is ok.
integer = LinkDest.getKind()    Get number of LinkDest kind.
string = LinkDest.getKindName()    Get name of LinkDest kind.
boolean = LinkDest.isPageRef()    TODO
integer = LinkDest.getPageNum()    TODO
Ref = LinkDest.getPageRef()    TODO
number = LinkDest.getLeft()    TODO
number = LinkDest.getBottom()    TODO
number = LinkDest.getRight()    TODO
number = LinkDest.getTop()    TODO
number = LinkDest.getZoom()    TODO
boolean = LinkDest.getChangeLeft()    TODO
boolean = LinkDest.getChangeTop()    TODO
boolean = LinkDest.getChangeZoom()    TODO
--- Links methods:
integer = Links.getNumLinks()    Get number of links.
Link = Links.getLink(integer)    Get link by number.
--- Object methods:
Object.initBool(boolean)    Initialize a Bool-type object.
Object.initInt(integer)    Initialize an Int-type object.
Object.initReal(number)    Initialize a Real-type object.
Object.initString(string)    Initialize a String-type object.
Object.initName(string)    Initialize a Name-type object.
Object.initNull()    Initialize a Null-type object.
Object.initArray(XRef)
    Initialize an A rray-type object with an empty array.
Object.initDict(XRef)
    Initialize a Dict-type object with an empty dictionary.
Object.initStream(Stream)    Initialize a Stream-type object.
Object.initRef(integer, integer)
    Initialize a Ref-type object by object number and generation.
Object.initCmd(string)    Initialize a Cmd-type object.
Object.initError()    Initialize an Error-type object.
Object.initEOF()    Initialize an EOF-type object.
Object = Object.fetch(XRef)
    If object is of type Ref, fetch and return the referenced object. Otherwise,
    return a copy of the object.
integer = Object.getType()
    Get object type as a number (enum ObjType).
string = Object.getTypeName()    Get object type name.
boolean = Object.isBool()    Check if object is of type Bool.
boolean = Object.isInt()    Check if object is of type Int.
boolean = Object.isReal()    Check if object is of type Real.
boolean = Object.isNum()    Check if object is of type Num.
boolean = Object.isString()    Check if object is of type String.
boolean = Object.isName()    Check if object is of type Name.
boolean = Object.isNull()    Check if object is of type Null.
boolean = Object.isArray()    Check if object is of type A rray.
boolean = Object.isDict()    Check if object is of type Dict.
boolean = Object.isStream()    Check if object is of type Stream.
boolean = Object.isRef()    Check if object is of type Ref.
boolean = Object.isCmd()    Check if object is of type Cmd.
boolean = Object.isError()    Check if object is of type Error.
boolean = Object.isEOF()    Check if object is of type EOF.
boolean = Object.isNone()    Check if object is of type None.
boolean = Object.getBool()    Get boolean from Bool-type object

```

```

integer = Object:getInt()    Get integer from Int-type object
number = Object:getReal()   Get number from Real-type object
number = Object:getNum()    Get number from Num-type object
string = Object:getString() Get string from String-type object
string = Object:getName()
    Get name from Name-type object as a string.
Array = Object:getArray()   Get Array from Array-type object
Dict = Object:getDict()     Get Dict from Dict-type object
Stream = Object:getStream() Get Stream from Stream-type object
Ref = Object:getRef()       Get Ref from Ref-type object
integer = Object:getRefNum()
    Get object number from Ref-type object
integer = Object:getRefGen()
    Get object generation from Ref-type object
string = Object:getCmd()    TODO
integer = Object:arrayGetLength()
    Get array length from Array-type object
Object:arrayAdd(Object)     Add Object to Array-type object
Object = Object:arrayGet(integer) Get Object from Array-type object
Object = Object:arrayGetNF(integer)
    Get Object from Array-type object, not resolving indirection.
integer = Object:dictGetLength(integer)
    Get dictionary length from Dict-type object
Object:dictAdd(string, Object) Add Object to Dict-type object
Object:dictSet(string, Object) TODO
Object = Object:dictLookup(string) TODO
Object = Object:dictLookupNF(string) TODO
string = Object:dictGetKey(integer)
    Get Dict key of Dict-type object by number.
Object = Object:dictGetVal(integer)
    Get Dict value of Dict-type object by number.
Object = Object:dictGetValNF(integer)
    Get Dict value of Dict-type object by number, not resolving indirection.
boolean = Object:streamIs(string)
    Check if object contains a stream whose dictionary is of given /Type.
Object:streamReset() TODO
integer = Object:streamGetChar() TODO
integer = Object:streamLookChar() TODO
integer = Object:streamGetPos() TODO
Object:streamSetPos(integer) TODO
Dict = Object:streamGetDict() TODO
--- Page methods
boolean = Page:isOk() Check if Page object is ok.
integer = Page:getNum() TODO
PDFRectangle = Page:getMediaBox() TODO
PDFRectangle = Page:getCropBox() TODO
boolean = Page:isCropped() TODO
number = Page:getMediaWidth() TODO
number = Page:getMediaHeight() TODO
number = Page:getCropWidth() TODO
number = Page:getCropHeight() TODO
PDFRectangle = Page:getBleedBox() TODO
PDFRectangle = Page:getTrimBox() TODO
PDFRectangle = Page:getArtBox() TODO
number = Page:getRotate() TODO
string = Page:getLastModified() TODO
Dict = Page:getBoxColorInfo() TODO
Dict = Page:getGroup() TODO
Stream = Page:getMetadata() TODO
Dict = Page:getPieceInfo() TODO
Dict = Page:getSeparationInfo() TODO
Dict = Page:getResourceDict() TODO
Object = Page:getAnnots() TODO
Links = Page:getLinks(Catalog) TODO
Object = Page:getContents() TODO
--- PDFDoc methods
boolean = PDFDoc:isOk() Check if PDFDoc object is ok.
integer = PDFDoc:getInt()    Get integer from Int-type object
number = PDFDoc:getReal()   Get number from Real-type object
number = PDFDoc:getNum()    Get number from Num-type object
string = PDFDoc:getString() Get string from String-type object
string = PDFDoc:getName()
    Get name from Name-type object as a string.
Array = PDFDoc:getArray()   Get Array from Array-type object
Dict = PDFDoc:getDict()     Get Dict from Dict-type object
Stream = PDFDoc:getStream() Get Stream from Stream-type object
Ref = PDFDoc:getRef()       Get Ref from Ref-type object
integer = PDFDoc:getRefNum()
    Get object number from Ref-type object
integer = PDFDoc:getRefGen()
    Get object generation from Ref-type object
string = PDFDoc:getCmd()    TODO
integer = PDFDoc:arrayGetLength()
    Get array length from Array-type object
Object:arrayAdd(Object)     Add Object to Array-type object
Object = PDFDoc:arrayGet(integer) Get Object from Array-type object
Object = PDFDoc:arrayGetNF(integer)
    Get Object from Array-type object, not resolving indirection.
integer = PDFDoc:dictGetLength(integer)
    Get dictionary length from Dict-type object
Object:dictAdd(string, Object) Add Object to Dict-type object
Object:dictSet(string, Object) TODO
Object = PDFDoc:dictLookup(string) TODO
Object = PDFDoc:dictLookupNF(string) TODO
string = PDFDoc:dictGetKey(integer)
    Get Dict key of Dict-type object by number.
Object = PDFDoc:dictGetVal(integer)
    Get Dict value of Dict-type object by number.
Object = PDFDoc:dictGetValNF(integer)
    Get Dict value of Dict-type object by number, not resolving indirection.
boolean = PDFDoc:streamIs(string)
    Check if object contains a stream whose dictionary is of given /Type.
Object:streamReset() TODO
integer = Object:streamGetChar() TODO
integer = Object:streamLookChar() TODO
integer = Object:streamGetRawChar() TODO
integer = Object:streamGetUnfilteredChar() TODO
Stream:unfilteredReset() TODO
integer = Stream:getPos() TODO
boolean = Stream:isBinary() TODO
Stream = Stream:getUndecodedStream() TODO
Dict = Stream:getDict() TODO
--- XRef methods
boolean = XRef:isOk() Check if XRef object is ok.
integer = XRef:getInt()    Get integer from Int-type object
number = XRef:getReal()   Get number from Real-type object
number = XRef:getNum()    Get number from Num-type object
string = XRef:getString() Get string from String-type object
string = XRef:getName()
    Get name from Name-type object as a string.
Array = XRef:getArray()   Get Array from Array-type object
Dict = XRef:getDict()     Get Dict from Dict-type object
Stream = XRef:getStream() Get Stream from Stream-type object
Ref = XRef:getRef()       Get Ref from Ref-type object
integer = XRef:getRefNum()
    Get object number from Ref-type object
integer = XRef:getRefGen()
    Get object generation from Ref-type object
string = XRef:getCmd()    TODO
integer = XRef:arrayGetLength()
    Get array length from Array-type object
Object:arrayAdd(Object)     Add Object to Array-type object
Object = XRef:arrayGet(integer) Get Object from Array-type object
Object = XRef:arrayGetNF(integer)
    Get Object from Array-type object, not resolving indirection.
integer = XRef:dictGetLength(integer)
    Get dictionary length from Dict-type object
Object:dictAdd(string, Object) Add Object to Dict-type object
Object:dictSet(string, Object) TODO
Object = XRef:dictLookup(string) TODO
Object = XRef:dictLookupNF(string) TODO
string = XRef:dictGetKey(integer)
    Get Dict key of Dict-type object by number.
Object = XRef:dictGetVal(integer)
    Get Dict value of Dict-type object by number.
Object = XRef:dictGetValNF(integer)
    Get Dict value of Dict-type object by number, not resolving indirection.
boolean = XRef:streamIs(string)
    Check if object contains a stream whose dictionary is of given /Type.
Object:streamReset() TODO
integer = Object:streamGetChar() TODO
integer = Object:streamLookChar() TODO
integer = Object:streamGetRawChar() TODO
integer = Object:streamGetUnfilteredChar() TODO
Stream:unfilteredReset() TODO
integer = Stream:getPos() TODO
boolean = Stream:isBinary() TODO
Stream = Stream:getUndecodedStream() TODO
Dict = Stream:getDict() TODO
--- XRef methods
boolean = XRef:isOk() Check if XRef object is ok.
integer = XRef:getInt()    Get integer from Int-type object
number = XRef:getReal()   Get number from Real-type object
number = XRef:getNum()    Get number from Num-type object
string = XRef:getString() Get string from String-type object
string = XRef:getName()
    Get name from Name-type object as a string.
Array = XRef:getArray()   Get Array from Array-type object
Dict = XRef:getDict()     Get Dict from Dict-type object
Stream = XRef:getStream() Get Stream from Stream-type object
Ref = XRef:getRef()       Get Ref from Ref-type object
integer = XRef:getRefNum()
    Get object number from Ref-type object
integer = XRef:getRefGen()
    Get object generation from Ref-type object
string = XRef:getCmd()    TODO
integer = XRef:arrayGetLength()
    Get array length from Array-type object
Object:arrayAdd(Object)     Add Object to Array-type object
Object = XRef:arrayGet(integer) Get Object from Array-type object
Object = XRef:arrayGetNF(integer)
    Get Object from Array-type object, not resolving indirection.
integer = XRef:dictGetLength(integer)
    Get dictionary length from Dict-type object
Object:dictAdd(string, Object) Add Object to Dict-type object
Object:dictSet(string, Object) TODO
Object = XRef:dictLookup(string) TODO
Object = XRef:dictLookupNF(string) TODO
string = XRef:dictGetKey(integer)
    Get Dict key of Dict-type object by number.
Object = XRef:dictGetVal(integer)
    Get Dict value of Dict-type object by number.
Object = XRef:dictGetValNF(integer)
    Get Dict value of Dict-type object by number, not resolving indirection.
boolean = XRef:streamIs(string)
    Check if object contains a stream whose dictionary is of given /Type.
Object:streamReset() TODO
integer = Object:streamGetChar() TODO
integer = Object:streamLookChar() TODO
integer = Object:streamGetRawChar() TODO
integer = Object:streamGetUnfilteredChar() TODO
Stream:unfilteredReset() TODO
integer = Stream:getPos() TODO
boolean = Stream:isBinary() TODO
Stream = Stream:getUndecodedStream() TODO
Dict = Stream:getDict() TODO

```

Font table

`metrics = font.read_tfm(string, number)`

Parse a font metrics file, at the size indicated by the number.

`metrics = font.read_vf(string, number)`

Parse a virtual font metrics file, at the size indicated by the number.

`metrics = font.getfont(number)` Fetch an internal font id as a lua table.

`font.setfont(number, metrics)` Set an internal font id from a lua table.

`boolean = font.frozen(number)`

True if the font is frozen and can no longer be altered.

`number = font.define(metrics)`

Process a font metrics table and stores it in the internal font table, returning its internal id.

`number = font.nextid()` Return the next free font id number.

`number = font.id(string)`

Return the font id of the font accessed by the csname given.

`[number] = font.current([number])` Get or set the currently active font

`number = font.max()` Return the highest used font id at this moment

`number, metrics = font.each()` Iterate over all the defined fonts

Font loader table

`table = fontloader.info(string)`

Get various information fields from an font file.

`fontloader.info` returned information:

key	type	explanation
<code>fontname</code>	string	the POSTSCRIPT name of the font
<code>fullname</code>	string	the formal name of the font
<code>familyname</code>	string	the family name this font belongs to
<code>weight</code>	string	a string indicating the color value of the font
<code>version</code>	string	the internal font version
<code>italicangle</code>	float	the slant angle

`luafont, table = fontloader.open(string, [string])`

Parse a font file and return a table representing its contents. The optional argument is the name of the desired font in case of font collection files.

The optional return value contains any parser error strings.

Listing all of the substructure returned from `fontloader.open` would take too much room, see the big reference manual.

`fontloader.apply_featurefile(luafont, string)`

Apply a feature file to a fontloader table.

`fontloader.apply_afmfile(luafont, string)`

Apply an AFM file to a fontloader table.

Image table

Full list of <image> object fields:

field name	type	description
<code>depth</code>	number	the image depth for L ^A T _E X (in scaled points)
<code>height</code>	number	the image height for L ^A T _E X (in scaled points)
<code>width</code>	number	the image width for L ^A T _E X (in scaled points)
<code>transform</code>	number	the image transform, integer number 0..7
<code>attr</code>	string	the image attributes for L ^A T _E X
<code>filename</code>	string	the image file name
<code>stream</code>	string	the raw stream data for an /XObject /Form object
<code>page</code>	??	the identifier for the requested image page (type is number or string, default is the number 1)
<code>pagebox</code>	string	the requested bounding box, one of none, media, crop, bleed, trim, art
<code>bbox</code>	table	table with 4 boundingbox dimensions llx, lly, urx, and ury overruling the pagebox entry
<code>filepath</code>	string	the full (expanded) file name of the image
<code>colordepth</code>	number	the number of bits used by the color space
<code>colorspace</code>	number	the color space object number

<code>imagetype</code>	string	one of pdf, png, jpg, jbig2, or nil
<code>objnum</code>	number	the PDF image object number
<code>index</code>	number	the PDF image name suffix
<code>pages</code>	number	the total number of available pages
<code>xsize</code>	number	the natural image width
<code>ysize</code>	number	the natural image height
<code>xres</code>	number	the horizontal natural image resolution (in DPI)
<code>yres</code>	number	the vertical natural image resolution (in DPI)

`image = img.new([table])`

This function creates an 'image' object. Allowed fields in the table: "filename" (required), "width", "depth", "height", "attr", "page", "pagebox", "colorspace".

`table = img.keys()`

Returns a table with possible image table keys, including retrieved information.

`image = img.scan(image)`

Processes an image file and stores the retrieved information in the image object.

`image = img.copy(image)` Copy an image.

`image = img.write(image)` Write the image to the PDF file.

`image = img.immediatewrite(image)`

Write the image to the PDF file immediately.

`node = img.node(image)` Returns the node associated with an image.

`table = img.types()` Returns a list of supported image types.

`table = img.bboxes()`

Returns a list of supported image bounding box names.

Kpathsea table

`kpse.set_program_name(string, [string])`

Initialize the kpathsea library by setting the program name. The optional string allows explicit program setting.

`kpathsea = kpse.new(string, [string])`

Create a new kpathsea library instance. The optional string allows explicit program setting.

`string = kpse.find_file(string, [string], [boolean], [number])`

Find a file. The optional string is the file type as supported by the standalone kpsewhich program (default is "tex", no autodiscovery takes place). The optional boolean indicates whether the file must exist. The optional number is the dpi value for PK files.

`string = kpse.lookup(string, table)` Find a file (extended interface).

The `kpse.lookup` options match commandline arguments from kpsewhich:

key	type	description
<code>debug</code>	number	set debugging flags for this lookup
<code>format</code>	string	use specific file type (see list above)
<code>dpi</code>	number	use this resolution for this lookup; default 600
<code>path</code>	string	search in the given path
<code>all</code>	boolean	output all matches, not just the first
<code>must-exist</code>	boolean	search the disk as well as ls-R if necessary
<code>mktxpk</code>	boolean	disable/enable mktxpk generation for this lookup
<code>mktextex</code>	boolean	disable/enable mktextex generation for this lookup
<code>mktxmf</code>	boolean	disable/enable mktxmf generation for this lookup
<code>mktextfm</code>	boolean	disable/enable mktextfm generation for this lookup
<code>subdir</code>	string or table	only output matches whose directory part ends with the given string(s)

`kpse.init_prog(string, number, string, [string])`

Initialize a PK generation program. The optional string is the metafont mode fallback name

`string = kpse.readable_file(string)`

Returns true if a file exists and is readable.
`string = kpse.expand_path(string)` Expand a path.
`string = kpse.expand_var(string)` Expand a variable.
`string = kpse.expand_braces(string)` Expand the braces in a variable.
`string = kpse.show_path(string)`
 List the search path for a specific file type.
`string = kpse.var_value(string)` Return the value of a variable.
`string = kpse.version()` Return the kpathsea version.

Language table

`language = lang.new([number])`
 Create a new language object, with an optional fixed id number.
`number = lang.id(language)`
 Returns the current internal language id number.
`[string] = lang.hyphenation(language, [string])`
 Get or set hyphenation exceptions.
`lang.clear_hyphenation(language)`
 Clear the set of hyphenation exceptions.
`string = lang.clean(string)`
 Creates a hyphenation key from the supplied hyphenation exception.
`[string] = lang.patterns(language, [string])`
 Get or set hyphenation patterns.
`lang.clear_patterns(language)` Clear the set of hyphenation patterns.
`[number] = lang.prehyphenchar(language, [number])`
 Set the pre-hyphenchar for implicit hyphenation.
`[number] = lang.posthyphenchar(language, [number])`
 Set the post-hyphenchar for implicit hyphenation.
`[number] = lang.preexhyphenchar(language, [number])`
 Set the pre-hyphenchar for explicit hyphenation.
`[number] = lang.postexhyphenchar(language, [number])`
 Set the post-hyphenchar for explicit hyphenation.
`boolean = lang.hyphenate(node, [node])` Hyphenate a node list

Lua table

There are 65536 bytecode registers, that are saved in the format file. Assignments are always global.

`function = lua.getbytecode(number)`
 Return a previously stored function from a bytecode register.
`lua.setbytecode(number, function)`
 Save a function in a bytecode register.
 They also be accessed via the virtual array `lua.bytecode[]`.
 The virtual array `lua.name[]` can be used to give names to lua chunks. To use
`lua.name[1]`, set `lua.name[1] = 'testname'` and `'directlua1{rubbish}'`.

Metapost table

`string = mplib.version()` Returns the mplib version.
`mpinstance = mplib.new(table)` Create a new metapost instance.
`mpdata = mp.execute(string)` Execute metapost code in the instance.
`mpdata = mp.finish()` Finish a metapost instance.
 The return value of `mp.execute` and `mp.finish` is a table with a few possible keys (only status is always guaranteed to be present).

<code>log</code>	<code>string</code>	output to the 'log' stream
<code>term</code>	<code>string</code>	output to the 'term' stream
<code>error</code>	<code>string</code>	output to the 'error' stream (only used for 'out of memory')
<code>status</code>	<code>number</code>	the return value: 0=good, 1=warning, 2=errors, 3=fatal error
<code>fig</code>	<code>table</code>	an array of generated figures (if any)

Handling of fig objects would take too much room here, please see the big reference manual.

`table = mp.statistics()`
 Returns some statistics for this metapost instance.
`number = mp.char_width(string, number)` Report a character's width.
`number = mp.char_height(string, number)`
 Report a character's height.
`number = mp.char_depth(string, number)` Report a character's depth.

Node table

`table = node.types()` Return the list of node types.
`table = node.whatsits()` Return the list of whatsit types.
`boolean = node.is_node(any)` Return true if the object is a <node>.
`number = node.id(string)`
 Convert a node type string into a node id number.
`number = node.subtype(string)`
 Convert a whatsit type string into a node subtype number.
`string = node.type(number)`
 convert a node id number into a node type string.
`table = node.fields(number, [number])`
 Report the fields a node type understands. The optional argument is needed for whatsits.
`boolean = node.has_field(node, string)`
 Return true if the node understands the named field.
`node = node.new(number, [number])`
 Create a new node with id and (optional) subtype.
`node.free(node)` Release a node.
`node.flush_list(node)` Release a list of nodes.
`node = node.copy(node)` Copy a node.
`node = node.copy_list(node, [node])` Copy a node list.
`node, number = node.hpack(node, [number], [string], [string])`
 Pack a node list into a horizontal list. The number is the desired size, the first string is either "exact" or "additional" (modifies the first string), the second string is the desired direction.
`node, number = node.vpack(node, [number], [string], [string])`
 Pack a node list into a vertical list. Arguments as for `node.hpack`.
`number, number, number = node.dimensions([number], [number], [number], node, [node])`
 Return the natural dimensions of a (horizontal) node list. The 3 optional numbers represent `glue_set`, `glue_sign`, and `glue_order`. The calculation stops just before the optional node (default end of list).
`node = node.mlist_to_hlist(node, string, boolean)`
 Recursively convert a math list into a horizontal list. The string differentiates display and inline, the boolean whether penalties are inserted.
`node = node.slide(node)`
 Move to the last node of a list while fixing next and prev pointers.
`node = node.tail(node)` Return the last node in a list.
`number = node.length(node, [node])`
 Return the length of a node list. Processing stops just before the optional node.
`number = node.count(number, node, [node])`
 Return the count of nodes with a specific id in a node list. Processing stops just before the optional node.
`node = node.traverse(node)` Iterate over a node list.
`node = node.traverse_id(number, node)`
 Iterate over nodes with id matching the number in a node list.
`node, node = node.remove(node, node)`
 Extract and remove a second node from the list that starts in the first node.
`node, node = node.insert_before(node, node, node)`
 Insert the third node just before the second node in the list that starts at the first node.
`node, node = node.insert_after(node, node, node)`
 Insert the third node just after the second node in the list that starts at the first node.
`node = node.first_glyph(node, [node])`
 Return the first character node in a list. Processing stops just before the optional node.

`node, node, boolean = node.ligaturing(node, [node])`
 Apply the internal ligaturing routine to a node list. Processing stops just before the optional node.

`node, node, boolean = node.kerning(node, [node])`
 Apply the internal kerning routine to a node list. Processing stops just before the optional node.

`node.unprotect_glyphs(node)`
 Mark all characters in a node list as being processed glyphs.

`node.protect_glyphs(node)`
 Mark all processed glyphs in a node list as being characters.

`node = node.last_node()`
 Pops and returns the last node on the current output list.

`node.write(node)`
 Appends a node to the current output list.

`boolean = node.protrusion_skippable(node)`
 Return true if the node could be skipped for protrusion purposes.

`node = node.next(node)`
 Returns the next node.

`node = node.prev(node)`
 Returns the previous node.

`number = node.has_attribute(node, number, [number])`
 Return an attribute value for a node, if it has one. The optional number tests for a specific value.

`node.set_attribute(node, number, number)`
 Set an attribute value for a node.

`number = node.unset_attribute(node, number, [number])`
 Unset an attribute value for a node. The optional number tests for a specific value.

Pdf table

`number = pdf.immediateobj([number], [string], string, [string])`
 Write an object to the PDF file immediately. The optional number is an object id, the first optional string is "file", "stream", or "filestream". the second optional string contains stream attributes for the latter two cases.

`pdf.mapfile(string)`
 Register a font map file.

`pdf.mapline(string)`
 Register a font map line.

`number = pdf.obj([number], [string], string, [string])`
 Write an object to the PDF file. See "pdf.immediateobj" for arguments.

`pdf.refobj(number)`
 Reference an object, so that it will be written out.

`number = pdf.pageref(number)`
 Return the pageref object number.

`pdf.print([string], string)`
 Write directly to the PDF file (use in `\atelu`). The optional string is one of "direct" or "page".

`number = pdf.reserveobj()`
 Reserve an object number in the PDF backend.

`pdf.registerannot(number)`
 Register an annotation in the PDF backend.

Status table

`table = status.list()`
 Returns a table with various status items.

The current list is:

key	explanation
<code>pdf_gone</code>	written PDF bytes
<code>pdf_ptr</code>	not yet written PDF bytes
<code>dvi_gone</code>	written DVI bytes
<code>dvi_ptr</code>	not yet written DVI bytes
<code>total_pages</code>	number of written pages
<code>output_file_name</code>	name of the PDF or DVI file
<code>log_name</code>	name of the log file
<code>banner</code>	terminal display banner
<code>var_used</code>	variable (one - word) memory in use
<code>dyn_used</code>	token (multi - word) memory in use
<code>str_ptr</code>	number of strings
<code>init_str_ptr</code>	number of INITEX strings
<code>max_strings</code>	maximum allowed strings
<code>pool_ptr</code>	string pool index

<code>init_pool_ptr</code>	INITEX string pool index
<code>pool_size</code>	current size allocated for string characters
<code>node_mem_usage</code>	a string giving insight into currently used nodes
<code>var_mem_max</code>	number of allocated words for nodes
<code>fix_mem_max</code>	number of allocated words for tokens
<code>fix_mem_end</code>	maximum number of used tokens
<code>cs_count</code>	number of control sequences
<code>hash_size</code>	size of hash
<code>hash_extra</code>	extra allowed hash
<code>font_ptr</code>	number of active fonts
<code>max_in_stack</code>	max used input stack entries
<code>max_nest_stack</code>	max used nesting stack entries
<code>max_param_stack</code>	max used parameter stack entries
<code>max_buf_stack</code>	max used buffer position
<code>max_save_stack</code>	max used save stack entries
<code>stack_size</code>	input stack size
<code>nest_size</code>	nesting stack size
<code>param_size</code>	parameter stack size
<code>buf_size</code>	current allocated size of the line buffer
<code>save_size</code>	save stack size
<code>obj_ptr</code>	max PDF object pointer
<code>obj_tab_size</code>	PDF object table size
<code>pdf_os_ptr</code>	max PDF object stream pointer
<code>pdf_os_objidx</code>	PDF object stream index
<code>pdf_dest_names_ptr</code>	max PDF destination pointer
<code>dest_names_size</code>	PDF destination table size
<code>pdf_mem_ptr</code>	max PDF memory used
<code>pdf_mem_size</code>	PDF memory size
<code>largest_used_mark</code>	max referenced marks class
<code>filename</code>	name of the current input file
<code>inputid</code>	numeric id of the current input
<code>lineinumber</code>	location in the current input file
<code>lasterrorstring</code>	last error string
<code>lua_bytecodes</code>	number of active LUA bytecode registers
<code>lua_bytecode_bytes</code>	number of bytes in LUA bytecode registers
<code>lua_state_bytes</code>	number of bytes in use by LUA interpreters
<code>output_active</code>	true if the output routine is active
<code>callbacks</code>	total number of executed callbacks so far
<code>indirect_callbacks</code>	number of those that were themselves a result of other callbacks (e.g. file readers)
<code>luatex_svn</code>	the luatex repository id (added in 0.51)
<code>luatex_version</code>	the luatex version number (added in 0.38)
<code>luatex_revision</code>	the luatex revision string (added in 0.38)
<code>ini_version</code>	true if this is an INITEX run (added in 0.38)

Typesetting table

`tex.set([string], string, value)`
 Set a named internal register. Also accepts a predefined csname string.

`value = tex.get(string)`
 Get a named internal register. Also accepts a predefined csname string.

Many of `LUATEX`'s internal parameters can be queried and set this way, but not nearly all. The big reference manual has an extensive list.

`tex.setattribute([string], number, number)`
 Set an attribute register. Also accepts a predefined csname string.

`number = tex.getattribute(number)`
 Get an attribute register. Also accepts a predefined csname string.

`tex.setbox([string], number, node)`
 Set a box register. Also accepts a predefined csname string.

`node = tex.getbox(number)`
 Get a box register. Also accepts a predefined csname string.

`tex.setcount([string], number, number)`
 Set a count register. Also accepts a predefined csname string.

`number = tex.getcount(number)`
 Get a count register. Also accepts a predefined csname string.

`tex.setdimen([string], number, number)`

Set a dimen register. Also accepts a predefined csname string.

`number = tex.getdimen(number)`

Get a dimen register. Also accepts a predefined csname string.

`tex.setskip([string], number, node)`

Set a skip register. Also accepts a predefined csname string.

`node = tex.getskip(number)`

Get a skip register. Also accepts a predefined csname string.

`tex.settoks([string], number, string)`

Set a toks register. Also accepts a predefined csname string.

`string = tex.gettoks(number)`

Get a toks register. Also accepts a predefined csname string.

`tex.setcatcode([string], [number], number, number)`

Set a category code.

`number = tex.getcatcode([number], number)` Get a category code.

`tex.setlccode([string], number, number, [number])`

Set a lowercase code.

`number = tex.getlccode(number)` Get a lowercase code.

`tex.setsfcode([string], number, number)` Set a space factor.

`number = tex.getsfcode(number)` Get a space factor.

`tex.setuccode([string], number, number, [number])`

Set an uppercase code.

`number = tex.getuccode(number)` Get an uppercase code.

`tex.setmathcode([string], number, table)` Set a math code.

`table = tex.getmathcode(number)` Get a math code.

`tex.setdelcode([string], number, table)` Set a delimiter code.

`table = tex.getdelcode(number)` Get a delimiter code.

In all the `tex.set...` functions above, the optional string is the literal "global".

The items can also be accessed directly via virtual arrays: `tex.attributes[]`,

`tex.box[]`, `tex.count[]`, `tex.dimen[]`, `tex.skip[]`, `tex.toks[]`; `tex.catcode[]`, `tex.lccode[]`,

`tex.sfcodes[]`, `tex.uccode[]`, `tex.mathcode[]`, `tex.delcode[]`.

`tex.setmath([string], string, string, number)`

Set an internal math parameter. The first string is like the csname but without the `\math` prefix, the second string is a style name minus the style suffix.

`number = tex.getmath(string, string)`

Get an internal math parameter. The first string is like the csname but without the `\math` prefix, the second string is a style name minus the style suffix.

`tex.print([number], string, [string])`

Print a sequence of strings (not just two) as lines. The optional argument is a catcode table id.

`tex.sprint([number], string, [string])`

Print a sequence of strings (not just two) as partial lines. The optional argument is a catcode table id.

`tex.tprint(table, [table])`

Combine any number of `tex.sprint`'s into a single function call.

`tex.write(string)`

Print a sequence of strings (not just two) as detokenized data.

`number = tex.round(number)` Round a number.

`number = tex.scale(number, number)`

Multiplies the first number (or all fields in a table) with the second argument (if the first argument is a table, so is the return value).

`number = tex.sp(string)` Convert a dimension string to scaled points

`tex.definefont([boolean], string, number)`

Define a font csname. The optional boolean indicates for global definition, the string is the csname, the number is a font id.

`tex.error(string, [table])`

Create an error that is presented to the user. The optional table is an array of help message strings.

`tex.enableprimitives(string, table)`

Enable the all primitives in the array using the string as prefix.

`table = tex.extraprimtives(string, [string])`

Return all primitives in a (set of) extension identifiers. Valid identifiers are: "tex", "core", "etex", "pdf_{te}x", "omega", "aleph", and "luat_{ex}".

`table = tex.primitives()`

Returns a table of all currently active primitives, with their meaning.

`number = tex.badness(number, number)` Compute a badness value.

`tex.linebreak(node, table)`

Run the line breaker on a node list. The table lists settings.

The `tex.linebreak` parameters:

name	type	description
<code>pardir</code>	string	
<code>pretolerance</code>	number	
<code>tracingparagraphs</code>	number	
<code>tolerance</code>	number	
<code>looseness</code>	number	
<code>hyphenpenalty</code>	number	
<code>exhyphenpenalty</code>	number	
<code>pdfadjustspacing</code>	number	
<code>adjdemerits</code>	number	
<code>pdfprotrudechars</code>	number	
<code>linepenalty</code>	number	
<code>lastlinefit</code>	number	
<code>doublehyphendemerits</code>	number	
<code>finalhyphendemerits</code>	number	
<code>hangafter</code>	number	
<code>interlinepenalty</code>	number or table	if a table, then it is an array like <code>\interlinepenalties</code>
<code>clubpenalty</code>	number or table	if a table, then it is an array like <code>\clubpenalties</code>
<code>widowpenalty</code>	number or table	if a table, then it is an array like <code>\widowpenalties</code>
<code>brokenpenalty</code>	number	
<code>emergencystretch</code>	number	in scaled points
<code>hangindent</code>	number	in scaled points
<code>hsize</code>	number	in scaled points
<code>leftskip</code>	glue_spec node	
<code>rightskip</code>	glue_spec node	
<code>pdfeachlineheight</code>	number	in scaled points
<code>pdfeachlinedepth</code>	number	in scaled points
<code>pdfastlineheight</code>	number	in scaled points
<code>pdfastlinedepth</code>	number	in scaled points
<code>pdfignoreddimen</code>	number	in scaled points
<code>parshape</code>	table	

The `tex.linebreak` returned table data:

<code>prevdepth</code>	depth of the last line in the broken paragraph
<code>prevgraf</code>	number of lines in the broken paragraph
<code>looseness</code>	the actual looseness value in the broken paragraph
<code>demerits</code>	the total demerits of the chosen solution

`tex.shipout(number)`

Ships the box to the output file and clears the box.

The virtual table `tex.lists` contains the set of internal registers that keep track of building page lists

field	description
<code>page_ins_head</code>	circular list of pending insertions
<code>contrib_head</code>	the recent contributions
<code>page_head</code>	the page-so-far
<code>hold_head</code>	used for held-over items for next page
<code>adjust_head</code>	head of the current <code>\adjust</code> list
<code>pre_adjust_head</code>	head of the current <code>\adjust pre</code> list

The virtual table `tex.nest` contains the currently active semantic nesting state. It has two main parts: an zero-based array of userdata for the semantic nest itself, and the numerical value `tex.nest.ptr`. Known fields:

key	type	modes	explanation
<code>mode</code>	number	all	The current mode. 0 = no mode, 1 = vertical, 127 = horizontal, 253 = display math. -1 = internal vertical, -127 = restricted horizontal, -253 = inline math.
<code>modeline</code>	number	all	source input line where this mode was

			entered in, negative inside the output routine.	index	meaning	description
head	node	all	the head of the current list	1	command code	this is a value between 0 and 130
tail	node	all	the tail of the current list	2	command modifier	this is a value between 0 and 2 ²¹
prevgraf	number	vmode	number of lines in the previous paragraph	3	control sequence id	for commands that are not the result of control sequences, like letters and characters, it is zero, otherwise, it is a number pointing into the 'equivalence table'
prevdepth	number	vmode	depth of the previous paragraph			
spacefactor	number	hmode	the current space factor			
dirs	node	hmode	internal use only			
noad	node	mmode	internal use only			
delimptr	node	mmode	internal use only			
mathdir	boolean	mmode	true when during math processing the \mathdir is not the same as the surrounding \textdir			
mathstyle	number	mmode	the current \mathstyle			

Texconfig table

This is a table that is created empty. A startup LUA script could fill this table with a number of settings that are read out by the executable after loading and executing the startup file.

key	type	default	explanation
kpse_init	boolean	true	false totally disables KPATHSEA initialisation
shell_escape	string		cf. web2c docs
shell_escape_commands	string		cf. web2c docs
string_vacancies	number	75000	cf. web2c docs
pool_free	number	5000	cf. web2c docs
max_strings	number	15000	cf. web2c docs
strings_free	number	100	cf. web2c docs
nest_size	number	50	cf. web2c docs
max_in_open	number	15	cf. web2c docs
param_size	number	60	cf. web2c docs
save_size	number	4000	cf. web2c docs
stack_size	number	300	cf. web2c docs
dvi_buf_size	number	16384	cf. web2c docs
error_line	number	79	cf. web2c docs
half_error_line	number	50	cf. web2c docs
max_print_line	number	79	cf. web2c docs
hash_extra	number	0	cf. web2c docs
pk_dpi	number	72	cf. web2c docs
trace_file_names	boolean	true	false disables T _E X's normal file feedback
file_line_error	boolean	false	file:line style error messages
halt_on_error	boolean	false	abort run on the first encountered error
formatname	string		if no formatname was given on the commandline, this will be used
jobname	string		as formatname.

IO table

texio.write([string], string)
Write a string to the log and/or terminal. The optional argument is "term", "term and log", or "log".

texio.write_nl([string], string)
Write a string to the log and/or terminal, starting on a new line. The optional argument is "term", "term and log", or "log".

Token table

A token is represented in LUA as a small table. For the moment, this table consists of three numeric entries:

```
token = token.get_next()  Fetch the next token from the input stream.
boolean = token.is_expandable(token)
    True if the token is expandable.
token.expand()
    Expand a token the tokenb waiting in the input stream.
boolean = token.is_activechar(token)
    True if the token represents and active character.
token = token.create(number, [number])
    Create a token from scratch, the optional argument is a category code.
    Also accepts strings, in which case a token matching that csname is created.
string = token.command_name(token)
    Return the internal string representing a command code.
number = token.command_id(string)
    Return the internal number representing a command code.
string = token.csname_name(token)
    Return the csname associated with a token.
number = token.csname_id(string)
    Returns the value for a csname string.
```

