

Unicode Bidi Algorithm Implementation for OpTeX, L^AT_EX, and Plain

Version: 1.1, 2026-09-18

Udi Fogiel, 2025-2026

The `unibidi-lua` LuaTeX package is an implementation of the [Unicode Annex #9](#) for OpTeX, L^AT_EX and Plain LuaTeX formats. The core Lua module also works with `texlua` and `plain lua`. It allows typesetting bidirectional documents without the need of special markup.

1 Usage

To use the package, as with other packages, you can do `\load[unibidi-lua]`, `\usepackage{unibidi-lua}` or `\input unibidi-lua` if you are using OpTeX, L^AT_EX or Plain respectively. The process function is added to the `pre_shaping_filter` when you load the package.

To modify things you can use the `\unibidilua` macro, which accepts key-value pairs separated by a space.

The macro accepts the following keywords:

- **enable:** This key accepts a boolean value. When true, the `unibidi-lua` process function is active in the `pre_shaping_filter` callback. When false, the callback remains registered but processing is skipped. Default is true when the package is loaded.
- **fences:** This key accepts a boolean value. It allows to disable or enable to consider matched brackets when resolving directional levels ([rule N0](#)). Default is true when the package is loaded.
- **nsm:** This key accepts a boolean value. It allows to disable or enable reordering of combining marks after the reordering of the text ([rule L3](#)). This key is only relevant for the string functions, as there is no reordering in the nodes functions (dir nodes are used there instead). Default is true when the package is loaded.
- **mirror:** This key accepts a boolean value. It allows to disable or enable mirroring of characters ([rule L4](#)). Default is true when the package is loaded.
- **linereset:** This key accepts a boolean value. When true, a `post_linebreak_filter` pass applies [rule L1](#) clause four on each line: it moves the trailing whitespace of every line out of the inner directional runs, so it renders in the paragraph embedding direction, at the visual end of the line. Because the node path resolves the paragraph as a single line, without this the trailing whitespace of an interior line keeps the direction of the run it ends, which can misplace it in bidirectional text. This key is only relevant for the node path (the string functions already apply [rule L1](#) per line, given the line breaks). It is on by default.
- **remove:** This key accepts one of `none`, `controls`, or `full`. This controls which characters are removed during processing. With `none`, nothing is removed, with `controls` only LRE, RLE, LRO, RLO, PDF characters will be removed, and with `full` all characters specified in [rule X9](#) are removed.
- **setdir:** This key accepts an integer (or a range of integers) representing Unicode code points, and a direction value. It sets the directional property for the specified character(s). Example: `setdir `A 1` or `setdir `A-`Z al`.
- **setmirror:** This key accepts an integer (or a range of integers) representing Unicode code points, and mirror data. It sets the mirroring property for the specified character(s), indicating what character they should mirror to in RTL contexts. Example: `setmirror `(`\)`.

- **setbracket:** This key accepts an integer (or a range of integers) representing Unicode code points, and a bracket type value (o (open) or c (close)) . It sets the bracket type property for the specified character(s). Example: `setbracket` \{ o`.
- **baselevel:** This key accepts a Lua function that determines the base directionality level of the text. See `bidi.set("baselevel")` in [section 2](#).
- **mirrorchar:** This key accepts a Lua function that applies the mirroring on nodes. See `bidi.set("mirrorchar")` in [section 2](#).
- **startlevel:** This key accepts a Lua function that determines from which level to insert direction nodes. See `bidi.set("startlevel")` in [section 2](#).

Note that you can also load the T_EX side interface without loading the package file using

```
\directlua{require('unibidi-lua-interface')}
```

This sets up the T_EX interface macros but does not add any function to a callback, and does not ensure `luaotfload` is loaded. The return value is a table with a `reorder` function and a `resetlines` function that can be added to callbacks manually:

```
\directlua{
  local unibidi = require('unibidi-lua-interface')
  luatexbase.add_to_callback("pre_shaping_filter", unibidi.reorder, "unibidi-lua")
  luatexbase.add_to_callback("post_linebreak_filter", unibidi.resetlines, "unibidi-lua")
}
```

2 Lua API

If you just want the Lua API, you can load it directly using

```
local bidi = require('unibidi-lua')
```

This does not define the T_EX interface nor add a function to a callback, and works with `texlua` and plain `lua` as well as LuaT_EX. Note that the node functions are only available when running under LuaT_EX.

2.1 Options

- `bidi.set(key, value)`: Set an option. The following keys are accepted:
 - `"fences"`: boolean, enable/disable bracket pair resolution ([rule N0](#)).
 - `"nsm"`: boolean, enable/disable combining mark reordering ([rule L3](#)). Only relevant for string processing functions.
 - `"mirror"`: boolean, enable/disable character mirroring ([rule L4](#)).
 - `"remove"`: string, one of `"none"`, `"controls"`, or `"full"`. Controls which characters are removed during processing.
 - `"baselevel"`: function, a custom function to determine the base directionality level of the paragraph. See the source code for more details.
 - `"mirrorchar"`: function, called for each glyph that has a mirror character. Receives the direct node and the mirror codepoint, and is responsible for applying the mirroring. The default implementation sets the character to its mirror for all non HarfBuzz fonts, as HarfBuzz handles mirroring (including using OpenType features). Note that this does not check OpenType mirroring features (`rtlm/rtla`); fonts that rely on those features for mirroring may not render correctly in node mode¹. Only relevant for node processing functions.

¹ If you want you can implement it using this hook. See for example <https://github.com/latex3/luatex3/blob/bidi-dev/src/luatex3-load-mirror.lua>

- **"startlevel"**: function, called with the base level to determine the minimum embedding level from which direction nodes are inserted. The default returns the base level itself.

For example consider the following

```
\hbox bdir1 {\r 42}
```

The list of nodes we start with is something like

```
glyph "Ṗ"
glyph "ṙ"
glue  " "
glyph "4"
glyph "2"
```

With the default **baselevel** function, the bidi algorithm is run with base level 1 (the direction of the hbox containing the text). The resolved levels are 1 1 1 2 2. Direction nodes are then inserted starting from **startlevel** to mark the boundaries of each directional run. With the default **startlevel** function the resulting node list is:

```
dir    +TRT
glyph  "Ṗ"
glyph  "ṙ"
glue    " "
dir    +TLT
glyph  "4"
glyph  "2"
dir    -TLT
dir    -TRT
```

The default inserts direction nodes starting from the base level, ensuring that the logical reading order is preserved if the box is later unboxed (since unboxing discards the box direction), or if a custom **baselevel** function assigns a different level than the containing box direction. If unboxing and custom base level functions are not a concern, the outer direction nodes can be omitted by returning **baselevel+1**:

```
bidi.set("startlevel", function(baselevel) return baselevel+1 end)
```

which produces:

```
glyph  "Ṗ"
glyph  "ṙ"
glue    " "
dir    +TLT
glyph  "4"
glyph  "2"
dir    -TLT
```

Only relevant for node processing functions.

- **bidi.get(key)**: Get the current value of an option. Accepts the same keys as **bidi.set**.

2.2 Data tables

- **bidi.directions**: Table containing character directional properties. Indexed by Unicode code point, returns the bidi class string (e.g. "l", "r", "al", "an", etc.).
- **bidi.mirrors**: Table containing character mirroring mappings. Indexed by Unicode code point, returns the code point of the mirrored character.
- **bidi.brackets**: Table containing bracket type classifications. Indexed by Unicode code point, returns "o" for opening brackets, "c" for closing brackets.

2.3 String functions

- `bidi.string.reorder(str, direction, where)`: Apply the Unicode Bidirectional Algorithm to the string `str` and return the visually reordered string. `direction` is 0 for LTR, 1 for RTL, or -1 (or `nil`) for auto-detect (or it can mean something else if the base level function is replaced). `where` is just a value that is passed to the base level function, and the default one does not use it.
- `bidi.string.levels(str, direction, where)`: Apply the Unicode Bidirectional Algorithm to the string `str` and return two values: a levels array and a reorder array. The levels array maps each character position to its resolved bidi level, or `false` if the character was removed by [rule X9](#). The reorder array maps each visual position to the original logical index of the character.
- `bidi.codepoints.reorder(codepoints, direction, where)`: Same as `bidi.string.reorder` but accepts and returns an array of codepoints instead of a string.
- `bidi.codepoints.levels(codepoints, direction, where)`: Same as `bidi.string.levels` but accepts an array of codepoints instead of a string.

The functions above treat the whole input as a single line. When the text is to be broken into several lines, the per-line rules [rule L1](#) (resetting separators and trailing whitespace to the base level) and [rule L2](#) (reordering) must be applied on each line, at line boundaries that are usually only known after shaping and line breaking. Since the base level resolution (rules [rule P2](#) through [rule I2](#)) is independent of the line breaks (and of glyph widths), it can be computed once and reused. This is what the paragraph objects are for.

- `bidi.string.paragraph(str, direction, where)`: Resolve the embedding levels of `str` (rules [rule P2](#) through [rule I2](#)) and return a paragraph object. The `direction` and `where` arguments are as in `bidi.string.reorder`. Unlike the flat functions above, a paragraph object owns its own storage and remains valid across later calls into the module, so it can be reused for several line breakings. The object provides two methods, both taking an optional `linebreaks` argument:
 - `paragraph.reorder(linebreaks)`: Apply [rule L1](#) and [rule L2](#) on each line and return the visually reordered string.
 - `paragraph.levels(linebreaks)`: Return a levels array and a reorder array, as in `bidi.string.levels`, but with [rule L1](#) applied on each line.
- `linebreaks` is an array of line ends, each the position one past the last character of a line, so line `k` spans the characters from the previous entry (or 1) up to `linebreaks[k]-1`. The values must be strictly increasing, and the last must equal the length of the text plus one. When `linebreaks` is omitted (or `nil`), the whole text is treated as a single line, so that `paragraph.reorder()` matches `bidi.string.reorder(str, direction, where)`.
- `bidi.codepoints.paragraph(codepoints, direction, where)`: Same as `bidi.string.paragraph` but accepts an array of codepoints; the resulting object's `:reorder` returns an array of codepoints instead of a string.

The options affecting level resolution and mirroring (`fences` and `mirror`) are applied when the object is constructed, while `nsm` and `remove` take effect when `:reorder` or `:levels` is called. Change these options before constructing the paragraph if in doubt.

2.4 Node functions

There are two variants of the node functions: `bidi.direct` which operates on direct node references (as used in `node.direct`), and `bidi.node` which operates on regular node references.

- `bidi.direct.reorder(head, direction, where)`: Apply the Unicode Bidirectional Algorithm to the node list starting at `head` (a direct node reference). Direction nodes are inserted to mark directional runs. Returns the (possibly new) head of the node list.
- `bidi.direct.levels(head, direction, where)`: Apply the Unicode Bidirectional Algorithm to the node list starting at `head` (a direct node reference) and return a levels array. The levels array maps each node position to its resolved bidi level, or `false` if the node was removed by [rule X9](#).

- `bidirect.node.reorder(head, direction, where)`: Same as `bidirect.direct.reorder` but accepts and returns regular node references.
- `bidirect.node.levels(head, direction, where)`: Same as `bidirect.direct.levels` but accepts regular node references.
- `bidirect.direct.resetlines(head)`: Apply rule [rule L1](#) clause four to a list of lines (as passed to `post_linebreak_filter`), `head` being a direct node reference. For each line it moves the trailing whitespace out of the inner directional runs, keeping it inside the base level scope, so it renders in the paragraph embedding direction. It relies on the base level tagged onto the paragraph node by `bidirect.direct.reorder`.
- `bidirect.node.resetlines(head)`: Same as `bidirect.direct.resetlines` but accepts and returns regular node references.