

Logical Order vs Visual Order  
stored                      normal display

Characters:

Strong

Weak

Neutral

Strong Characters: Letters in alphabet, each with a direction

LTR      most alphabets

RTL      Arabic + Persian, Urdu; Hebrew; and others

A change in direction creates a directional boundary

From now on, in examples (**sans serif**), lower case letters are LTR strong and **UPPER CASE LETTERS** are RTL strong

Left column shows logical order and right column shows visual order

Unless otherwise stated, document direction is LTR

ab	ab
AB	BA
aABbc	aBAbc
AabBC	AabCB

abAB	abBA
	BAab

if document direction is RTL

Weak Characters: digits and other punctuation and symbols that appear with numerals, e.g., . , ° \$

LTR

RTL

When weak characters are embedded inside other text, they do not create a directional boundary EVEN when they are embedded in text of the opposite direction

This comes from the fact that in Arabic + and Hebrew, numerals are written from left to right but are considered not as a shift to LTR text as would embedded English would be.

ab12c

ab12c

A12BC

CB12A (compare with AabBC)

Neutral Characters: white space, international punctuation not appearing inside or with a numeral

A neutral character takes its direction from its surroundings,  
e.g.:

Neutral character  $c$  surrounded by strong characters of the same direction  $d \Rightarrow c$  gets direction  $d$ .

aaaAB!Cbbb    aaaC!BAbbb

! is RTL since it is surrounded by two RTL strong characters **B** and **C**

Neutral character *c* surrounded by strong characters of different directions  $\Rightarrow$  *c* gets direction of document.

aaaAB!bbb      aaaBA!bbb

! is LTR since it is surrounded by an RTL and an LTR strong character, and the document direction is LTR

One way to force ! to be RTL is to use [RLM]

Each of [LRM] (LTR Mark) and [RLM] (RTL Mark) is zero width, strong, and directed as the first two letters of its name.

aaaAB!<sub>[RLM]</sub>bbb      aaa!BAbbb

! is RTL since it is surrounded by two RTL strong characters B and [RLM]

Aside: also the space is neutral. So, it gets treated the same way as the !.

ab\_cd\_AB\_CD\_ef\_gh

ab→cd→AB←CD→ef→gh    ab cd DC BA ef gh

Each of the first and last →s is LTR since it is surrounded by two strong LTR letters. The ← in the middle is RTL since it is surrounded by two strong RTL letters. Each of the inner →s is LTR since the letters that surround it are of different directions, but the document direction is LTR.

Other ways to achieve the same visual ordering:

Embedding

Overriding

Embedding:

[LRE] to the next [PDF]

[RLE] to the next [PDF]

([LRE], [RLE] are strong, [PDF] is weak, and each is zero width)

Embedding is used to mark a block of text as a subdocument with its own direction.

(In Unicode, each paragraph is a document whose direction is determined by its first strong character, which could be a letter, [LRM], [RLM], [LRE], [RLE], [LRO], or [RLO])

In an embedding, a neutral character gets the direction of the embedding.

aaa<sub>[RLE]</sub>AB!<sub>[PDF]</sub>bbb      aaa!BAbbb

! is RTL since it is surrounded by an RTL and an LTR strong character, and the subdocument (embedding) direction is RTL.

Embedding is not recommended as are [RLM] and [LRM] since one can forget to close off an embedding with [PDF] (but maybe the word processor can enforce that!).

Overriding:

[LRO] to the next [PDF]

[RLO] to the next [PDF]

([LRO], [RLO] are strong, [PDF] is weak, and each is zero width)

Within an override, a neutral character, and for that matter a strong or weak character, is forced to be strong in the override direction.

aaa<sub>[RLO]</sub>AB!<sub>[PDF]</sub>bbb      aaa!BAbbb

! is RTL since it has been forced to be RTL by being in an RTL overriding.

Overriding is not recommended as are [RLM] and [LRM] since one can forget to close off an overriding with [PDF] (but maybe the word processor can enforce that!).

Notice the difference between embedding and overriding, with respect to the treatment of weak characters.

abA12B!cd            abB12A!cd



abA12B!<sub>[RLM]</sub>cd        ab!<sub>B</sub>12Acd

ab<sub>[RLE]</sub>A12B!<sub>[PDF]</sub>cd    ab!<sub>B</sub>12Acd

ab<sub>[RLO]</sub>A12B!<sub>[PDF]</sub>cd    ab!<sub>B</sub>21Acd

Now some tables directly from the Unicode Bidi Algorithm Specification in Lucida Sans Unicode (with an occasional comment in Times Roman):



## 2.1 Explicit Directional Embedding

Abbr.	Code	Chart	Name	Description
LRE	U+202A		LEFT-TO-RIGHT EMBEDDING	Treat the following text as embedded left-to-right.
RLE	U+202B		RIGHT-TO-LEFT EMBEDDING	Treat the following text as embedded right-to-left.

The effect of right-left line direction, for example, can be accomplished by embedding the text with RLE...PDF.

## 2.2 Explicit Directional Overrides


The following codes allow the bidirectional character types to be overridden when required for special cases, such as for part numbers.

Abbr	Code	Char	Name	Description
.		t		
LRO	U+202D		LEFT-TO-RIGHT OVERRIDE	Force following characters to be treated as strong left-to-right characters.
RLO	U+202E		RIGHT-TO-LEFT OVERRIDE	Force following characters to be treated as strong right-to-left characters.



The right-to-left override, for example, can be used to force a part number made of mixed English, digits and Hebrew letters to be written from right to left.

## 2.3 Terminating Explicit Directional Code

The following code terminates the effects of the last explicit code (either embedding or override) and restores the bidirectional state to what it was before that code was encountered.

Abbr	Code	Char Name	Description
.		t	
PDF	U +202C	 POP DIRECTIONAL FORMATTING	Restore the bidirectional state to what it was before the last LRE, RLE, RLO, or LRO.

## 2.4 Implicit Directional Marks

Abbr.	Code	Char Name	Description
LRM	U+200E	 LEFT-TO-RIGHT MARK	Left-to-right zero-width character
RLM	U+200F	 RIGHT-TO-LEFT MARK	Right-to-left zero-width character

... their effect on bidirectional ordering is exactly the same as a corresponding strong directional character; the only difference is that they do not appear in the display.

## 3.2 Bidirectional Character Types

**Table 4.** Bidirectional Character Types

Category	Type	Description	General Scope
Strong	L	Left-to-Right	LRM, most alphabetic, syllabic, Han ideographs, non-European or non-Arabic digits, ...
	LRE	Left-to-Right Embedding	LRE
	LRO	Left-to-Right Override	LRO
	R	Right-to-Left	RLM, Hebrew alphabet, and related punctuation
	AL	Right-to-Left Arabic	Arabic, Thaana, and Syriac alphabets, most punctuation specific to those scripts, ... (these may connect, Hebrew does not!)
	RLE	Right-to-Left Embedding	RLE
	RLO	Right-to-Left Override	RLO

Weak	PDF	Pop Directional Format	PDF
	EN	European Number	European digits, Eastern Arabic-Indic digits, ...
	ES	European Number Separator	<b>plus sign, minus sign</b>
	ET	European Number Terminator	<b>degree sign</b> , currency symbols, ...
	AN	Arabic Number	Arabic-Indic digits, Arabic decimal and thousands separators, ...
	CS	Common Number Separator	<b>colon, comma, full stop</b> (period), <b>no-break space</b> , ...
	NSM	Nonspacing Mark	Characters marked Mn (Nonspacing_Mark) and Me (Enclosing_Mark) in the Unicode Character Database (These are Arabic and Hebrew vowels or diacritics.)
	BN	Boundary Neutral	Default ignorables, non-characters, and control characters, other than those explicitly given other types. (These are control characters after they have been obeyed once, to prevent second obeying.)

Neutral	B	Paragraph Separator	<b>paragraph separator</b> , appropriate Newline Functions, higher-level protocol paragraph determination
	S	Segment Separator	Tab
	WS	Whitespace	<b>space, figure space, line separator, form feed</b> , General Punctuation spaces, ...
	ON	Other Neutrals	All other characters, including <b>object replacement character</b> (and ES, ET, and CS characters that are not parts of ENs and ANs)

## Summary of Unicode Bidi Algorithm as User Sees It:

A Unicode compliant word processor is supposed to do the EFFECT of this algorithm after EACH change to the logically ordered file.

Each step is a complete pass over the characters  $c$  in the file, from beginning to end.

1. Break the text into paragraphs. For each paragraph  $p$ , give  $p$  an initial current embedding level (CEL) according to its first strong character,  $c$ .

if  $c$  is LTR then  $CEL \leftarrow 0$  else ( $c$  is RTL)  $CEL \leftarrow 1$

(Thus each paragraph gets its own document direction based on its first character. Thus, if you want a paragraph's direction to be different from that of its first character, insert LRM or RLM before its first character to force the direction the other way.)

## 2. First assignment of embedding level (EL):

In a complete pass over the characters  $c$  in the file, from beginning to end:

if  $c = \text{LRE}$  or  $c = \text{LRO}$  then push CEL;

CEL  $\leftarrow$  next higher even level

(0,1  $\rightarrow$  2; 2,3  $\rightarrow$  4; etc)

if  $c = \text{RLE}$  or  $c = \text{RLO}$  then push CEL;

CEL  $\leftarrow$  next higher odd level

(0  $\rightarrow$  1; 1,2  $\rightarrow$  3; 3,4  $\rightarrow$  5 etc)

EL( $c$ )  $\leftarrow$  CEL

if c = PDF then pop CEL

(to recover CEL = what it was before the corresponding  
LRE, LRO, RLE, RLO)

(Note that a direction change among strong and weak characters does NOT change the CEL. So, if there is never a LRE, LRO, RLE, RLO, or PDF, then each neutral gets the direction of the first strong character in its paragraph.)

3. Apply each override to all characters of any strength in its embedding level until and including its corresponding PDF, forcing each character to have the direction of its override. (An override is interrupted for a nested embedding level.)

4. Give a direction and possibly a new type to each character of weak type.

(Remember that weak type characters are intended to be numerals, including their punctuation, which are written LTR even in the midst of RTL text, such as European numbers (ENs), e.g., \$150.25, and Arabic numbers (ANs), e.g., \xi . . . . (No \$ in this number).)

A European separator character between two EN characters is considered an EN character unless the ENs immediately follow a strong Arabic letter (AL).

A Common separator character between two EN characters is considered an EN character.

A Common separator character between two AN characters is considered an AN character.

A European terminator character adjacent to an EN character is considered an EN character unless the ENs immediately follow a strong AL.

Otherwise each separator or terminator is considered a neutral character.

Finally, any sequence of ENs immediately following a strong LTR letter is considered a sequence of strong LTR letters.

5. Give a direction to each neutral character:

In a complete pass over the characters  $c$  in the file, from beginning to end:

Consider the characters  $b$  and  $d$ , before and after  $c$  in the file

If both  $b$  and  $d$  are strong and  
 $\text{direction}(b) = \text{direction}(d) = D$  then  
 $\text{direction}(c) \leftarrow D$

elseif  $b$  is strong and  $\text{direction}(b) = \text{RTL}$   
and  $d$  is an AN or EN then  
 $\text{direction}(c) \leftarrow \text{RTL}$

elseif  $b$  is an AN or EN and  $d$  is strong and  
 $\text{direction}(d) = \text{RTL}$  then  
 $\text{direction}(c) \leftarrow \text{RTL}$

elseif b is an AN or EN and d is an AN or EN then  
direction (c)  $\leftarrow$  RTL

else direction(c)  $\leftarrow$  direction(EL(c))

In the above,

if c is at the beginning of a run then

b is the boundary at the beginning of the run

if c is at the end of a run then

d is the boundary at the end of the run

a boundary is considered strong and

its direction is that of the higher EL

on the two sides of the boundary

6. Second assignment of embedding level (EL):

In a complete pass over the characters  $c$  in the file, from beginning to end:

if  $EL(c)$  is even (LTR) then

if  $c$  is a strong RTL letter then  $EL(c) \leftarrow EL(c) + 1$

elseif  $c$  is an AN or EN then  $EL(c) \leftarrow EL(c) + 2$

else ( $EL(c)$  is odd (RTL))

if  $c$  is a strong LTR or an AN or an EN then

$EL(c) \leftarrow EL(c) + 1$

(In this pass, within any run at any EL, each substring in the opposite direction gets a higher EL of the correct direction and within any run at an even (LTR) EL, each subsubstring which is an AN or EN in the same direction gets a still higher EL in the LTR direction, thus creating nested runs (fake embeddings) in preparation for the reversing of RTL text in Step 9.)

7. Carry out mirroring on mirror characters in odd ELs:

e.g., a "(" at an odd EL is changed to ")" and a ")" at an odd EL is changed to "(".

8. Form Arabic ligatures, replacing adjacent Arabic characters by a new one. Then determine each Arabic character's position and assign its shape.

(Now the width of each character is known and the length of sequences of characters can be calculated.)

9. Compute visual ordering of characters line by line:

For each line  $l$ ,

for  $i$  from the highest EL  $elh$  in  $l$   
to the lowest odd EL  $loel$  in  $l$ ,

for each extended run  $er$  of text in  $l$  of  
ELs  $i$  through  $elh$ ,

reverse the characters of  $er$  in place in  $l$ .

(This is a recursive version of the basic algorithm described in the “Brief Bi-Directional Text Reading Lesson”.)

in logical order with strengths, directions (A=Arabic + R), initial ELs, directions for neutrals, and final ELs:

```
dan lives at SALAAM 49AB15 SHALOM in a beautiful house.  
SSSNSSSSSNSSSNSSSSSSSNWWSSWNSSSSSSSNSSNSNSSSSSSSSSSSNSSSSSN  
LLL LLLLL LL AAAAAA nnRRnn RRRRRR LL L LLLLLLLLLL LLLLL  
000000000000000000000000000000000000000000000000000000000000  
  L      L L      R      R      L L L      L      L  
000000000000011111112211221111111000000000000000000000000000
```

reverse EL 2:

```
dan lives at SALAAM 94AB51 SHALOM in a beautiful house.
```

reverse ELs 1 + 2:

```
dan lives at MOLAHS 15BA49 MAALAS in a beautiful house.  
which is the desired visual order
```



with RLO ( $\Rightarrow$ ) and PDF( $\uparrow$ )

in logical order with strengths, directions (A=Arabic + R), initial ELs, directions for neutrals, and final ELs:

```
dan lives at  $\Rightarrow$ SALAAM 49AB15 SHALOM $\uparrow$  in a beautiful house.  
SSSNSSSSSNSSSNSSSSSSSNWWSSWWNSSSSSSWNSSNSNSSSSSSSSSNSSSSSN  
LLL LLLLL LL RRRRRRRRRRRRRRRRRRRRRRRR LL L LLLLLLLLL LLLLL  
0000000000000111111111111111111110000000000000000000000  
    L      L  L                      L  L  L                L      L  
000000000000011111111111111111111111110000000000000000000000
```

reverse EL 1 with RLO and PDF removed:

```
dan lives at MOLAHS 51BA94 MAALAS in a beautiful house.  
which is the desired visual order
```

Because a comma is not a European number terminator, in Step 4, they end up being neutral in the following examples. By the "else" of Step 5, the list of numbers separated by neutrals and embedded in a directional run will come out in the run's order.

**Storage:** he said "THE VALUES ARE 123, 456, 789, OK".

**Display:** he said "KO ,789 ,456 ,123 ERA SEULAV EHT".

In this case, both the comma and the space between the numbers take on the direction of the surrounding text (uppercase = right-to-left), ignoring the numbers. The commas are not considered part of the number because they are not surrounded on both sides by digits.

However, if there is a preceding left-to-right sequence, then European numbers will adopt that direction:

**Storage:** IT IS A bmw 500, OK.

**Display:** .KO ,bmw 500 A SI TI

The following examples illustrate the reordering, showing the successive steps in the application of Step 9. The original text, including any embedding codes for producing the particular levels, is shown in the "Storage" row in the examples. Each successive row thereafter shows the one pass of reversal from Step 9. At each iteration, the underlining shows the text that has been reversed.

The paragraph embedding level for the first and third examples is 0 (left-to-right direction), and for the second and fourth examples is 1 (right-to-left direction).

## Example 1 (embedding level = 0)

Storage: car means CAR.

Before Reordering: car means CAR.

Resolved levels: 00000000001110

Reverse level 1: car means RAC.

## Example 2 (embedding level = 1)

Storage:  car MEANS CAR. 



Before Reordering: car MEANS CAR.

Resolved levels: 22211111111111

Reverse level 2: rac MEANS CAR.

Reverse levels 1-2: .RAC SNAEM car

### Example 3 (embedding level = 0)

Storage: he said "  car MEANS CAR  ."





Before Reordering: he said "car MEANS CAR."

Resolved levels: 000000000222111111111100

Reverse level 2: he said "rac MEANS CAR."

Reverse levels 1-2: he said "RAC SNAEM car."

### Example 4 (embedding level = 1)

Storage: DID YOU SAY '  he said "  car MEANS CAR   '?"

Before Reordering: DID YOU SAY 'he said "car MEANS CAR" '?

Resolved levels:: 11111111111112222222244433333333211

Reverse level 4: DID YOU SAY 'he said "rac MEANS CAR" '?

Reverse levels 3-4: DID YOU SAY 'he said "RAC SNAEM car" '?

Reverse levels 2-4: DID YOU SAY ' "rac MEANS CAR" dias eh' ?

Reverse levels 1-4: ?'he said "RAC SNAEM car" ' YAS UOY DID

Because of the implicit character types and the heuristics for resolving neutral and numeric directional behavior, the implicit bidirectional ordering will generally produce the correct display without any further work. However, problematic cases may occur when a right-to-left paragraph begins with left-to-right characters, or there are nested segments of different-direction text, or there are weak characters on directional boundaries. In these cases, embeddings or directional marks may be required to get the right display. Part numbers may also require directional overrides.

The most common problematic case is that of neutrals on the boundary of an embedded language. This can be addressed by setting the level of the embedded text correctly. For example, with all the text at level 0 the following occurs:

**Memory:** he said "I NEED WATER!", and expired.

**Display:** he said "RETAW DEEN I!", and expired.

If the exclamation mark is to be part of the Arabic quotation, then the user can select the text I NEED WATER! and explicitly mark it as embedded Arabic, which produces the following result:

**Memory:** he said "<RLE>I NEED WATER!<PDF>", and expired.

**Display:** he said "!RETAW DEEN I", and expired.

However, a simpler and better method of doing this is to place a right directional mark (**RLM**) after the exclamation mark. Because the exclamation mark is now not on a directional boundary, this produces the correct result.

**Memory:** he said "I NEED WATER!**<RLM>**", and expired.

**Display:** he said "!RETAW DEEN I", and expired.

This latter approach is preferred because it does not make use of the stateful format codes, which can easily get out of sync if not fully supported by editors and other string manipulation.

The stateful format codes are generally needed only for more complex (and rare) cases such as double embeddings, as in the following:

**Memory:**

DID YOU SAY '<LRE>he said "I NEED WATER!<RLM>", and expired.<PDF>'?

**Display:**

? 'he said "!RETAW DEEN I", and expired.' YAS UOY DID