Bidi for Translators

Sasha Richalim (src4062@proton.me)

May 2, 2026

Abstract

This document is intended for translators working with bidirectional text in plain text format, primarily in the UI/UX domain. In this document, we will delve into the concepts of directionality, RTL and LTR text, and how they interact when combined. We will further understand how computers handle bidirectional text, the challenges they face, and how we can help them cope with their occasional shortcomings.

1 Directionality in Writing Systems

Individual writing systems make different assumptions about the arrangement of characters into lines and the arrangement of lines on a page or screen. We refer to this the writing system's **directionality**.

For example, letters are placed horizontally left-to-right in Latin to form lines. And lines of text are laid out running top to bottom on a page. Because the predominant directional flow of the Latin script is left to right, the Latin script is called "left-to-right".

Similarly, in some Middle Eastern scripts, all characters are arranged right to the left. However, more modern scripts like Arabic and Hebrew are more complex. The basic letters are right to left, while digits flow in the opposite direction.

Left-to-right:
Hello world!

Right-to-left:
مرحبا بالعالم!

2 Bidirectionality

When a text contains words of different scripts with varying directionality, it is called **bidirectional**, or **bidi** for short. Modern right-to-left scripts like Arabic and Hebrew combined with opposite directional scripts (like Latin to embed text) are examples of bidi. For example:

اللقاح COVID

Note that text is stored in **logical order**, that is, the order in which the letters are *pronounced*. Text is not stored in visual order. For the previous example, the order is as follows (the beginning of the order is from the left):

C O V I D <space> ا ل ل ق ا ح

3 Computers and Unicode

Computers need to accomplish two jobs, allowing both humans and themselves to understand these characters. This is where **character encoding** comes onto the stage. It is the process by which computers assign numbers to graphical characters, allowing them to be understood and transformed by computers.

This task of character encoding is primarily carried out by **Unicode**, or formally, **The Unicode Standard**. It is a character encoding standard maintained by the **Unicode Consortium**. It aimed to support the use of text in *all* the world's writing systems. Unicode, in the form of `UTF-8`, has been the most common encoding for the World Wide Web since 2008. As of 2024, 168 scripts are included in the latest version of Unicode, including many additional characters used in mathematics, music, and so on.

Unicode Encodings

The Unicode Standard itself defines three encodings: `UTF-8`, `UTF-16`, and `UTF-32`, though several others exist. Of these, `UTF-8` is the most widely used by a large margin, in part due to its backwards-compatibility with `ASCII` (an older encoding standard).

4 Referring to Unicode Characters

In order to refer to a particular Unicode character, we must keep in mind the concept of **code points**. In this notation, we prefix `U+` followed by a hexadecimal number. The range of possible code points in Unicode ranges from `U+0000` to `U+10FFFF`.

Hexadecimal Numbers

Hexadecimal (or "hex") is a number system that uses 16 digits instead of the usual 10 digits (0–9) we use in the decimal system. You don't have to worry about what those are. Just know that this allows us to represent large numbers using less digits. This aligns well with the way computers handle data.
To give an example, the range of `0x0000-0x10FFFF` (in hexadecimal) equals 0-1,114,111 in decimal. That means 1.1 million possible code points!

Consider the following examples:

- Code point for `DIVISION SIGN` (÷): `U+00F7`
- Code point for `GRINNING FACE` (😀): `U+1F600`
- Code point for `ARABIC LETTER BEH` (ب): `U+0628`

Knowing how to refer to Unicode characters will help us later when we use Unicode characters called **control characters** to help us solve some bidi issues. Please refer to the table in appendix A for detailed information of important control characters.

5 Unicode's Role in Bidi

Each Unicode character has a character property called `Bidi_Class` that indicates its directionality. Latin characters have a **strong left-to-right** directional type, while Arabic characters have a **strong right-to-left** directional type. There are also characters of **neutral** or **weak direction**. This includes most punctuation and symbols because either type of script may use them.

There are many Unicode characters have an alternate mirror form on the vertical axis. Such characters tend to be automatically reverse when used in RTL. For example:

a > b > c
ج < ب < ا

In both the examples, it is the *same* angle bracket. However, the glyph pair chosen is done so according to the prevailing direction. Most mirrored characters are paired. There are some standalones that inherit the right directionality despite not having a pair.

Note: "Smart" quotation marks and brackets are **not** flipped. You need to interchange their positions. For instance:

"Arabic"

when translated (only text), becomes:

"اَلْعَرَبِيَّةُ"

So we need to interchange the two characters to obtain:

"اَلْعَرَبِيَّةُ"

All of the above information is used by the Unicode Bidirectional Algorithm, defined by the Unicode Standard.

The Unicode Bidirectional Algorithm, often abbreviated to just UBA, explains in detail how text should be laid out in lines whenever it consists of a mixture of left-to-right script characters and right-to-left script characters. In some cases, the bidi algorithm copes fine with bidirectional text using its rules; in others, it needs some help. We shall discuss some basic rules to keep in mind and ways to fix problems if things go wrong. But first, we should be familiar with base direction and directional runs.

6 Base Direction

The order in which a text is displayed depends on the base direction or the directional context of the phrase, paragraph or block containing it. It establishes a directional context that the bidi algorithm refers to at various points to decide how to handle the text. This is crucial for directional runs in an overall block of text (see section 7).

Consider the following example:

ضد COVID انتقل

If the base direction was left-to-right instead, we would get:

انتقل COVID ضد

Thus, having the correct base direction is extremely important to set beforehand in order for things to be readable.

7 Directional Runs

When text with different directionality is mixed in a line, the Unicode bidi algorithm produces a separate "directional run" from each sequence of contiguous characters with the same directionality. These directional runs are like clumps of characters with the same directionality.

For instance, the below Arabic phrase has three directional runs:

ضد COVID انتقل

Here, "ضد" (RTL), "COVID" (LTR) and "انتقل" (RTL) are the directional runs.

The order in which these directional runs are displayed depends on the base direction assigned to the phrase, paragraph or block that contains it.

In case of neutral characters, they inherit the directionality of the surrounding text and *extend* its directionality based on certain rules, which will be discussed in later sections. Numbers are also weakly typed as they do not break the directional run. However, they run left to right within any text direction. Hence, there are only two directional runs in the below examples,

ثلاثة 1234 خمسة one two
ثلاثة ١٢٣٤ خمسة one two

8 Rules for Inherited Direction of Neutral Characters

When the UBA encounters characters with neutral directional type, it works out how to handle these by looking at the surrounding characters. Consider the following example:

The title is مفتاح معا يير الويب in Arabic.

8.1 Between strongly typed characters of same directionality

Rule: *A neutral character will inherit the same directionality if between two strongly typed characters that have the same directional type.*

For example:

The title is مفتاح!، معا يير الويب in Arabic.

So a neutral character between two RTL characters will be treated as a RTL character, and will have the effect of *extending* the directional run.

8.2 Between strongly typed characters of different directionality

Rule: *When a space or punctuation falls between two strongly typed characters that have different directionality, i.e., at the boundary of directional runs the neutral character inherits the same directionality as the surrounding directional context.*

Suppose the title in Arabic had an exclamation mark at the end. Adding it in logical order, we result with the following being displayed:

The title is مفتاح معا يير الويب! in Arabic.

Here, the higher directional context of the "!" mark is left-to-right. So, the above is expected, but this isn't our intention.

We have two ways of fixing this: (a) using `RLM` and (b) using a pair of characters called `RLI ... PDI`.

8.2.1 Using `RLM`

To change this, we can place the Unicode control character called `RLM` after the exclamation mark. This basically sandwiches the exclamation mark and makes its directional context right-to-left.

The title is مفتاح معا يير الويب!RLM in Arabic.
The title is مفتاح معا يير الويب! in Arabic.

8.2.2 Using `RLI ... PDI`

An alternative is to use a pair of characters called `RLI` and `PDI`. We will be discussing more about this pair of characters in the next section.

The title is RLIمفتاح معا يير الويب!PDI in Arabic.
The title is مفتاح معا يير الويب! in Arabic.

9 Embedding Directional Changes Using Bidi-Isolation

Now that we have understood the behaviour of strongly and weakly typed characters, we move on to the next essential aspect of bidi. If you want to insert a run of text as a unit while preserving its internal layout and not affecting the layout of the surrounding text, use **"bidi-isolation"** using the isolation control characters. Using isolation is the most common and safe way to ensure that each directional change is tightly wrapped and is, hence, appropriately displayed. The following sections discuss some common use cases of bidi-isolation.

It is worth noting that, although it is not required if the base direction and the embedded directional runs are properly enclosed, it is safer to enclose every directional change within a paragraph with the appropriate characters.

9.1 Simple Embedding

Consider the following example:

Organisation: W3C نشاط التدويل،

At first, it seems like adding `RLM` after W3C will solve the problem, but it doesn't, as it has a strong directional property. So, to fix the whole phrase, we need to enclose it using the control characters `RLI ... PDI`.

Organisation: RLI نشاط التدويل، W3C PDI
Organisation: W3C نشاط التدويل،

This tightly wraps each directional change.

9.2 Isolation in Lists

The names of these states in Arabic are مصر, البحرين and الكويت.

But "مصر" is first in the list. So we need to put the items within `RLI ... PDI` (or just "مصر" within it).

The names of these states in Arabic are RLI مصر PDI, RLI البحرين PDI and RLI الكويت PDI.
The names of these states in Arabic are مصر, البحرين and الكويت.

9.3 Isolation: Numbers

`${ title }` – `${ reviewTally }` reviews
تصميم و إنشاء موقع الوتب -- 4 reviews

Here the base direction is LTR. The `${ title }` variable can take book title names from various different languages (here it happens to be Arabic) and the `${ reviewTally }` takes a number. The text is displayed incorrectly because the 4 considers itself to be a part of the Arabic directional run. You could say that we can add `RLI` and `PDI` to the title and well that would work, but only if we knew beforehand that the title was RTL. But the title can be in any language. So instead, we need to use another control character – `FSI`.

تصميم و إنشاء موقع الوتب[FSI][PDI] – 4 reviews
تصميم و إنشاء موقع الوتب – 4 reviews

But it doesn't always work! Consider the title to be "HTML و CSS: تصميم و إنشاء مواقع الوى". Here there is not much we can do. We have to use `RLI ... PDI`. In such cases, we require metadata for correct handling.

Consider another example for isolating numbers:

المعرف: aa:4a:bb:06:01:02

Here, the MAC address is actually "01:02:aa:4a:bb:06". So, to fix this, we'll use `FSI ... PDI`.

المعرف: 01:02:aa:4a:bb:06
المعرف: [FSI]01:02:aa:4a:bb:06[PDI]

10 Helpful Tools

- r12a Unicode character apps: A web app allows you to produce, analyse and manipulate runs of text for a given language or script. There also options to list the characters in logical order and add control characters to text as well.
- Unicode Explorer: A website that allows you to search for unicode characters and copy them.
- View non-printable unicode characters: An online tool to display non-printable characters that may be hidden in copy and pasted strings.

A Table of Important Unicode Control Characters

Unicode provides control characters for when the Unicode Bidi Algorithm does not properly display text as the user intended.

Use one of the following characters to indicate the start of the embedded direction change.

Psst: Click on the name of the character to open its details on unicode-explorer.com.

Character	Name	Code Point	Comment
LRI	LEFT-TO-RIGHT ISOLATE	U+2066	Sets direction to LTR and isolates the embedded content from the surrounding text.
RLI	RIGHT-TO-LEFT ISOLATE	U+2067	Ditto, but for RTL.
FSI	FIRST-STRONG ISOLATE	U+2068	Isolates the content and sets the direction according to the first strongly typed directional character.

Then, you need to close the range with the following.

Character	Name	Code Point	Comment
PDI	POP DIRECTIONAL ISOLATE	U+2069	Used for RLI, LRI or FSI.

Unicode provides two other invisible format characters related to direction. They extend or break the ranges established by default by the bidi algorithm.

Character	Name	Code Point	Comment
LRM	LEFT-TO-RIGHT MARK	U+200E	Strongly typed LTR character
RLM	RIGHT-TO-LEFT MARK	U+200F	Strongly typed RTL character.

However, one thing these single characters cannot do is establish a base direction for an embedded range of inline text so that punctuation and nested direction changes are handled properly. For those use cases you need to use the paired characters.