

Minimum(ish) Working Example of Quarto with Arabic RTL text in an LTR document

The Authors

2024-11-16

Example watermark

© 2023 Author Names
All rights reserved.

This work may not be distributed or modified, without written permission from the copyright holder.

This book is typeset with Xe_{La}TeX (via Quarto and pandoc) in the Charis, Vazirmatn, and Amiri typefaces.

Example watermark

Table of contents

Preface	4
1 Introduction	5
1.1 Quick start guide	5
2 Arabic (العربية) support	7
2.1 In-line Arabic: spans	7
2.1.1 Arabic spans for HTML output	7
2.1.2 Arabic spans for PDF output	8
2.1.3 Rendered output of Arabic span	8
2.2 Arabic block text: divs	8
2.2.1 Arabic divs for HTML output	9
2.2.2 Arabic divs for PDF output	9
2.2.3 Rendered output of Arabic div	10
2.3 Pandoc Lua filters	10
2.4 Arabic fonts	13
2.4.1 Specify Arabic font for HTML	13
2.4.2 Specify Arabic font for PDF	14
3 Transliteration of Arabic	16
3.1 Romanization scheme	16
3.2 Fonts	18
3.3 Test transliteration 'abjd hwz ḥty klmn seṣṣ qrsṭh ṭḥkḥdh ḍḍḥgh āūī	18
4 Figures	19
5 Summary	23
5.1 Dummy section	23
References	24

Preface

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Quarto is a document publishing software. With it, you can write your document in **Pandoc flavored markdown**. Quarto will use Pandoc under the hood, and do a bunch of other fancy stuff, to output your markdown document in formats of your choice, like HTML for websites, and PDF (via **Latex**). This site is itself produced using Quarto.

So far so good. But many of my documents are in English with Arabic content interspersed. Arabic is written right-to-left (RTL) whereas English is written left-to-right (LTR). The support of bidirectional (BiDi) text is a notoriously tricky problem. The cursive property of the Arabic script (with joining letters) compounds the issue.

In this write-up, I will describe how to configure Quarto to solve some of these issues.

The source code for this book can be used as a template for RTL document projects in Quarto. Along with BiDi, I'll also discuss other aspects like fonts, figures, etc. **إن شاء الله**.

The source code for this book can be found here: <https://github.com/adamiturabi/quarto-arabic-mwe>.

The rendered output is published here: <https://adamiturabi.github.io/quarto-arabic-mwe/>.

A PDF version of this document can be downloaded if you click on the PDF icon next to the title at the top left of this page.

If you have any suggestions for improvements I'd love to know about them in the [discussions page](#) for this project.

1 Introduction

We assume familiarity with basic Quarto commands and project directory structure.

In the next chapter we will explain in detail how and why Quarto needs text to be input in order to render Arabic correctly. If you wish to avoid the technical discussion, and just want to know how to get going, follow the quick start guide below.

1.1 Quick start guide

In order to render Arabic text correctly in your project, use the source code for this book as a template.

First clone or download the repo from here: <https://github.com/adamiturabi/quarto-arabic-mwe>

You must have [Quarto](#) and the following fonts installed on your system:

- [Charis SIL](#)
- [Amiri](#)
- [Vazirmatn](#)

Edit or replace one or more of the `.qmd` files in the `srcqmd` directory directly with your text material. Make sure to update the list of `.qmd` files in the `_quarto.yml` file.

Arabic text is input with the following syntax:

Input for an Arabic (inline) span with sample contents:

```
[|ا هذا|]{.ar}
```

Input for an Arabic (block) div with sample contents:

```
 :::{.ar}  
 طويّل. عربي كلام ه ذا  
 سطرين. النص يبلّغ حتّى أكتب أنّ أريد  
 الخارجيّ. الملف لإنتاج قوارطو برنامج أستعمل  
 قبل. من أستعمله كنت الذي بكداؤن البرنامج خلف قد جيّد برنامج ه  
 :::
```

Rebuild the project with

```
quarto render
```

Example watermark

2 Arabic (العربية) support

There are two main ways to insert RTL Arabic text in an LTR document:

1. In-line Arabic within an LTR paragraph.
2. A block of Arabic text by itself

2.1 In-line Arabic: spans

For inline Arabic, we will use a Pandoc span. A span is written using this syntax:

```
[This is the span's *content text*]{.classname attributekey="attributeval"}
```

Within square brackets [] is the content of the span. This is what will be rendered in the output. Within the curly braces {} is a class name and some attributes that are needed by Quarto to properly process the span.

2.1.1 Arabic spans for HTML output

In order to render the Arabic content text correctly for HTML output, the span is input thus in the .qmd source file.

```
[العربي. نص هـا]{.reg-ar-txt dir="rtl" lang="ar"}
```

(Note that the the Arabic text in code listings (like the one above) does not render correctly in the PDF output, exemplifying how tricky BiDi support is. We haven't attempted to find a workaround for this.)

The class name is arbitrary. We suggest using a descriptive name. We will be using it in the CSS for selecting the font later. The output HTML code will be something like:

```
<span class="reg-ar-txt" dir="rtl" lang="ar">نص عربي. هذا</span>
```

2.1.2 Arabic spans for PDF output

For PDF output, the `dir="rtl"` attribute is unneeded, and in fact, clashes with the Xelatex PDF engine that Quarto mandates we use for documents with RTL text. So the span will need to be input thus in the `.qmd` source file:

```
[|ا هذا]{.reg-ar-txt lang="ar"}
```

The output Latex code will be something like:

```
\foreignlanguage{arabic}{ا هذا {عربي.}}
```

Under the hood, `\foreignlanguage` is a command that is used by the Latex package `babel` that Pandoc specifies in its Latex template for handling multiple languages and their scripts.

2.1.3 Rendered output of Arabic span

Finally, this is an example of an English sentence with inline Arabic text within it. Locate this sentence in the source code file [here](#) to see how we wrote it.

2.2 Arabic block text: divs

In order to write a block (paragraph) of Arabic text within an LTR document we will use a Pandoc div. A div is written using this syntax:

```
::{.classname attributekey="attributeval"}
This is the divs's *content text*.
:::
```


2.2.1 Arabic divs for HTML output

For HTML output, a div is input thus in the .qmd source:

```

:::{.reg-ar-txt dir="rtl" lang="ar"}
سطرين. النص يبلغ حتى أكتب أن أريد طويلاً. عربي كلام هذا
الخارجي. الملف لإنتاج قوارطو برنامج أستعمل
قبل. من أستعمله كنت الذي بكداؤن البرنامج خلف قد جيدي برنامج هو
:::

```

The class name `reg-ar-txt` is, again, arbitrary. The output HTML code will be:

```

<div class="reg-ar-text" lang="ar" dir="rtl">
طويلاً. عربي كلام هذا
سطرين. النص يبلغ حتى أكتب أن أريد
الخارجي. الملف لإنتاج قوارطو برنامج أستعمل
قبل. من أستعمله كنت الذي بكداؤن البرنامج خلف قد جيدي برنامج هو
</div>

```

2.2.2 Arabic divs for PDF output

For PDF output, a div is input thus in the .qmd source:

```

:::{.otherlanguage data-latex="{arabic}" lang='ar'}
طويلاً. عربي كلام هذا
سطرين. النص يبلغ حتى أكتب أن أريد
الخارجي. الملف لإنتاج قوارطو برنامج أستعمل
قبل. من أستعمله كنت الذي بكداؤن البرنامج خلف قد جيدي برنامج هو
:::

```

In this case, the class name `otherlanguage` is not arbitrary. Furthermore, another attribute `data-latex="{arabic}"` is also needed. And, as with spans, `lang="ar"` is needed but `dir="rtl"` should not be used. The output Latex code is:

```
\begin{otherlanguage}{arabic}
```

```
طويل. عربي كلام هذا
سطرين. النص يبلغ حتى أكتب أن أريد
الخارجي. الملف لإنتاج قوارطو برنامج أستعمل
قبل. من أستعمله كنت الذي بكداؤن البرنامج خلف قد جيد برنامج هو
```

```
\end{otherlanguage}
```

2.2.3 Rendered output of Arabic div

Finally, this is an example of an Arabic div. Locate it in the source code file [here](#) to see how we wrote it.

هذا كلام عربي طويل. أريد أن أكتب حتى يبلغ النص سطرين. أستعمل برنامج قوارطو لإنتاج الملف الخارجي. هو برنامج جيد قد خلف البرنامج بكداؤن الذي كنت أستعمله من قبل.

2.3 Pandoc Lua filters

As you can see, the process for typing Arabic text is both lengthy, and different for HTML and PDF outputs. In order to simplify it, we can use [Pandoc Lua filters](#).

We have created a Quarto filter extension (which is a grouping of Lua filters) to support Arabic divs and spans. The process for creating a Quarto filter extension is detailed here: <https://quarto.org/docs/extensions/filters.html>

This is the filter `inline-arabic-span.lua` that we wrote for handling Arabic spans:

```
-- Add attributes for Arabic text in a span
function Span (el)
  if el.classes:includes 'ar' or el.classes:includes 'aralt' then
    text = pandoc.utils.stringify(el)
    contents = {pandoc.Str(text)}
    if FORMAT:match 'latex' then
```

```

-- for handling alternate Arabic font
if el.classes:includes 'aralt' then
  -- can't seem to use string concatenate directly. Have to use RawInline
  table.insert(
    contents, 1,
    pandoc.RawInline('latex', '\\altfamily ')
  )
end
-- No dir needed for babel and throws error if it sees dir attribute.
-- It was previously needed for polyglossia
return pandoc.Span(contents, {lang='ar'})
elseif FORMAT:match 'html' then
  classval = 'reg-ar-text'
  if el.classes:includes 'aralt' then
    classval = 'alt-ar-text'
  end
  -- dir needed for html otherwise punctuation gets messed up
  return pandoc.Span(contents, {class=classval, lang='ar', dir='rtl'})
end
end
end

```

This is the filter `arabic-div.lua` that we wrote for handling Arabic divs:

```

-- Add attributes for Arabic text in a div
function Div (el)
  if el.classes:includes 'ar' or el.classes:includes 'aralt' then
    text = pandoc.utils.stringify(el)
    contents = {pandoc.Str(text)}
    if FORMAT:match 'latex' then
      -- for handling alternate Arabic font
      if el.classes:includes 'aralt' then
        -- can't seem to use string concatenate directly. Have to use RawInline
        table.insert(
          contents, 1,
          pandoc.RawInline('latex', '\\altfamily ')
        )
      end
      -- No dir needed for babel and throws error if it sees dir attribute.
      -- It was previously needed for polyglossia
    end
  end

```

```

return pandoc.Div(
  contents,
  {class='otherlanguage', data_latex="{arabic}", lang='ar'}
)
elseif FORMAT:match 'html' then
  classval = 'reg-ar-text'
  if el.classes:includes 'aralt' then
    classval = 'alt-ar-text'
  end
  -- dir needed for html otherwise punctuation gets messed up
return pandoc.Div(
  contents,
  {class=classval, lang='ar', dir='rtl'}
)
end
end
end

```

With activating these filters, now you can use Arabic divs and spans using a simplified syntax.

Input for an Arabic span:

```
[{.ar}{عربي. نص هـذا}]
```

Input for an Arabic div:

```

:::{.ar}
طويل. عربي كلام هـذا
سطرين. النص يبلغ حتى أكتب أن أريد
الخارجي. الملف لإنتاج قوارطو برنامج أستعمل
قبل. من أستعمله كنت الذي بكداؤن البرنامج خلف قد جيـد برنامج هو
:::

```

The filters will process them correctly for HTML and PDF output. Note that the class name `reg-ar-text` is hardcoded in the filter. If you wish to modify it you can edit the Lua files directly.

2.4 Arabic fonts

You can use a specific font for the Arabic text which is different from the font used for the English text. This is usually desirable because the typeface design for the Latin font often does not optimize (or even sometimes support) an Arabic font.

For my project, I am using the [Vazirmatn](#) and [Amiri](#) fonts.

Both of these are well designed fonts. For me, a major consideration is good typesetting of diacritics and the hamza character (ء). (See here for what I'm talking about: <https://adamiturabi.github.io/hamza-rules/#typographical-limitations>)

To specify the Arabic fonts, the process is different for HTML vs PDF output. We'll describe both below:

2.4.1 Specify Arabic font for HTML

For HTML output, the Arabic font is specified in the CSS file. The class name that we selected previously `reg-ar-text` is now assigned a font:

```
.reg-ar-text {  
  font-family: Vazirmatn, serif;  
  /* scaled up slightly w.r.t. the Latin font for readability */  
  font-size: 1.2em;  
  /* line spacing not scaled for visual congruence at the expense of clashes */  
  line-height: 100%;  
}
```

You will also need to add the font files to your project. Quarto will copy them over to the output directory so that they can be served to the browsers of visitors viewing your site. Be aware of fonts licences before uploading and using fonts in this way. Instead of uploading font files, you can instead use a font delivery service like Google Fonts, although they often have outdated versions. See [our fonts directory](#).

The font names `Vazirmatn` and `AmiriWeb` are defined in the same CSS file. A relative path to the font files is needed in the CSS file. See [our CSS file](#) for details.

In our CSS file, we have specified the font Amiri as an alternate font:

```
.alt-ar-text {
  font-family: AmiriWeb, serif;
  font-size: 1.2em;
}
```

It can be selected in the .qmd source with `{.aralt}` instead of `{.ar}`. You can also see how we handle it in the source code for the Lua filters above.

Here is an example of a div and a span in the alternate Arabic font. Span: هذا نص عربي.

Div:

هذا كلام عربي طويل. أريد أن أكتب حتى يبلغ النص سطرين. أستعمل برنامج قوارطو لإنتاج الملف الخارجي. هو برنامج جيد قد خلف البرنامج بكداؤن الذي كنت أستعمله من قبل.

By the way, I am not, by any means, an expert (or even proficient) in CSS, so if you see any problems with this method of specifying the font, feel free to let me know in the discussions or issues of the [Github page for this book](#).

2.4.2 Specify Arabic font for PDF

As we mentioned earlier, Latex uses the babel package to handle support for multiple languages. In order to specify Arabic font(s), we need to add the following lines in the intermediate .tex file produced by Quarto:

```
\babelfont[arabic]{rm}[Language=Default]{Vazirmatn-Light}
\babelfont[arabic]{sf}[Language=Default]{Vazirmatn-Light}
\babelfont[arabic]{alt}[Language=Default]{Amiri}
```

Quarto provides hooks for inserting such additional code using [includes](#) and [templates](#).

The above lines of code need to be inserted at a specific point after the `usepackage{babel}` line. We found that replacing the partial template for

`before-title.tex` worked in this case. Here is the addition in `our _quarto.yml` file:

```
format:
  pdf:
    template-partials:
      - src/ex/before-title.tex
```

Again, see our source code in Github for more details.

By the way, the fonts will need to have been installed on your system in order to generate the PDF output.

Example watermark

3 Transliteration of Arabic

3.1 Romanization scheme

In my documents, I frequently need to transliterate and transcribe Arabic text in Latin characters. There are various Romanization schemes in existence, using dots, macrons, etc. The Romanization scheme I am using for a few of my projects is tabulated below:

Arabic letter	Transliterated output	ASCII input
ء	'	E
ا	<i>ā</i>	A
ب	<i>b</i>	b
ت	<i>t</i>	t
ث	<i>th</i>	v
ج	<i>j</i>	j
ح	<i>h</i>	H
خ	<i>kh</i>	x
د	<i>d</i>	d
ذ	<i>ḍh</i>	p
ر	<i>r</i>	r
ز	<i>z</i>	z
س	<i>s</i>	s
ش	<i>sh</i>	c
ص	<i>ṣ</i>	S
ض	<i>ḍ</i>	D
ط	<i>ṭ</i>	T
ظ	<i>ḏh</i>	P
ع	<i>ε</i>	e
غ	<i>gh</i>	g
ف	<i>f</i>	f
ق	<i>q</i>	q

Arabic letter	Transliterated output	ASCII input
ك	<i>k</i>	k
ل	<i>l</i>	l
م	<i>m</i>	m
ن	<i>n</i>	n
ه	<i>h</i>	h
و (C/V)	<i>w/ū</i>	w/U
ي (C/V)	<i>y/ī</i>	y/I

As you can see, I use digraphs (\widehat{dh} , \widehat{gh} , etc.) for some letters. This is because, for my current projects, I prefer readability over precision.

It is possible to input these special characters directly by modifying your keyboard layout or mapping, either at an operating system, or editor level. Andreas Hallberg has described a technique for inputting them in the vim editor here: <https://andreasmhallberg.github.io/ergonomic-arabic-transcription/>

For Quarto, I prefer to input the transliterated text as ASCII characters. I have written a Lua filter `transliteration-span.lua` to handle rendering them correctly. The mapping of ASCII input to transliterated output is shown in the table above and is encoded in the filter. So if I input:

```
[pahabtu maphaban]{.trn}
```

It will be output as $\widehat{dh}ahabtu\ \widehat{madh}haban$.

Note the dot character `.` is automatically inserted by the filter between the digraph \widehat{dh} and the following *h* for helping in disambiguation.

With `{.trn}` the output is in italic (as above). But sometimes I need to have non-italic output, as in the case of names. For that I use `{.trn2}`. For example:

```
[#eAEicah]{.trn2} and [E#Adam]{.trn2} are studying  
the [#qurEAn]{.trn2} and [#HadIv]{.trn2}.
```

This is rendered as:

Ēā'ishah and 'Ādam are studying the Qur'ān and Ḥadīth.

Note how the hash character `#` is used to control capitalization.

3.2 Fonts

For the Latin font used in your main text, you will need to pick a font that supports the dots, macrons, breves, etc needed for transliteration. For my transliteration scheme, the font will also need to support U+02be for ’ and U+025b for ε. Not all Latin fonts support these extra characters. In this document, I am using the [Charis SIL](#) font.

Other fonts I have experimented with, that have varied support for these characters, are:

- [New Computer Modern](#)
- [DejaVu Serif](#)
- [Junicode](#)
- [Brill](#)
- [Gentium Plus](#)
- [STIX Two Text](#)

3.3 Test transliteration ’abjd hwz ḥṭy klmn sḥṣ qrsḥt thkhḏh ḏḏhgh āūī

Dummy text

4 Figures

I sometimes need to use TiKZ figures with Arabic text in my document. This chapter describes how to do that.

In order to process TiKZ code in Quarto I've modified a filter that was originally published here: https://github.com/danmackinlay/quarto_tikz

The filter needs some options we've set globally in `_quarto.yml`:

```
tikzarabic:
  cache: true
  format: svg
  embed_mode: link
  engine: lualatex
  template_html: srctex/tikz-template-html.tex
  template_pdf: srctex/tikz-template-pdf.tex
  libgs: /usr/local/Cellar/ghostscript/10.04.0/lib/libgs.dylib
```

Here is an explanation of the options:

- `cache`: Uses a cache on the system in order not have to evaluate the TiKZ code for every render. Hasn't been tested fully. Default value: `false`.
- `format`: SVG preserves vector information for optimal rendering. Default value: `svg`.
- `embed_mode`: Only affects HTML output. If set to `link` then the HTML includes the figure as a link. If set to `inline` then the HTML encodes the figure inline as ASCII text. Default value: `inline`.
- `engine`: The Arabic typesetting I'm using within the figures uses the package `arabluatex` which needs the LuaTeX engine. This is specified using the text `lualatex`. Ideally, I should use `xelatex` to be consistent with the rest of the document but I haven't yet figured out how to do

that. There may be some incompatibility between the babel package and TiKZ which would need to be resolved. Default value: `latex`.

- `template_html`: Common TiKZ template code for HTML output. Here's what I'm using:

```
\RequirePackage{luatex85}
\documentclass{article}
\usepackage[luatex,active,tightpage]{preview}
\usepackage{amsmath}
\usepackage{tikz}

\usepackage{fontspec}
\defaultfontfeatures{Scale=MatchLowercase}
\defaultfontfeatures[\rmfamily]{Ligatures=TeX,Scale=1}
\setmainfont{Charis SIL}

\usetikzlibrary{%s}
\usetikzlibrary{matrix}
\usepackage{arabxuatex}
\newfontfamily{\arabicfont}[Script=Arabic,Scale=1.2]{Vazirmatn-Regular}

\begin{document}
\begin{preview}
%s
\end{preview}
\end{document}
```

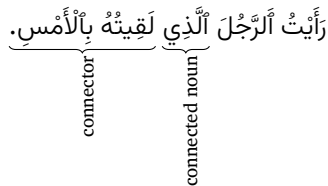
The code block in the `.qmd` file will be inserted in the second `%s`'s location. Additional TiKZ libraries may be inserted in the first `%s`'s location using the YAML option `libraries`.

Default template:

```
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\usetikzlibrary{%s}
\begin{document}
%s
\end{document}
]]
```

- `template_pdf`: Common TiKZ template code for PDF output. My template is identical to `template_html` except for a different weight for the Arabic font: “light” instead of “regular” to match the main document. The default template is the same as the default for `template_html`.
- `libs`: In order to use this filter for HTML output, you need to make sure that the utilities `dvisvgm` and `Ghostscript` are installed on your system. This can be verified using the commands: `which dvisvgm` and `which gs`. If `dvisvgm` was not compiled with `Ghostscript` then you may need to link the location of the Ghost script library on your system using the option `libs`. Default value: “”.

Here is an example of a TiKZ diagram with Arabic text:



“I saw the [specific] man whom I met yesterday.”

This is the source code for the figure in the [source .qmd file](#):

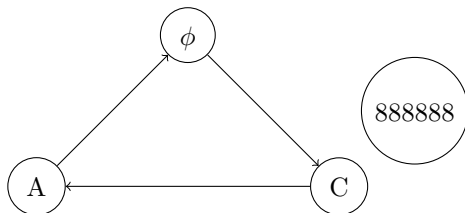
```
\usetikzlibrary{decorations.text, decorations.pathreplacing}
\begin{tikzpicture}[nodes={text depth=0.25ex, text height=2.0ex}]
\path [decoration={text effects along path,
  text=.\txarbo{لَقِيتُهُ بِالْأَمْسِ} \txarb{الَّذِي} \txarbo{رَأَيْتُ} \txarb{الرَّجُلَ},
  text effects/.cd,
  path from text, text along path,
  group letters, word count=\w,
  every word/.style={name=word-\w, execute at begin node=\strut}},
  decorate] (0,0);

\draw [decoration={brace, mirror}, decorate]
  (word-1.south west) -- (word-1.south east)
  node [midway, rotate=90, anchor=south, left=0ex, align=left] {\scriptsize connector};

\draw [decoration={brace, mirror}, decorate]
  (word-2.south west) -- (word-2.south east)
  node [midway, rotate=90, anchor=south, left=0ex, align=left] {\scriptsize connected noun};
```

```
\path (0,0) node [midway, below=13.5ex, right=0ex, align=left] {\`I saw the [specific] man whom
\end{tikzpicture}
```

If you wish to override any of the global parameters you can do some inline using the comment-pipe (%%| option: value) format. Here is an example:



Source code:

```
%%| embed_mode: inline
%%| engine: default
%%| template_html: default
%%| template_pdf: default
\begin{tikzpicture}[nodes={text depth=0.25ex,text height=2.0ex}]
\node[draw, circle] (A) at (0,0) {A};
\node[draw, circle] (B) at (2,2) {$\phi$};
\node[draw, circle] (C) at (4,0) {C};
\node[draw, circle] (seven) at (5,1) {888888};
\draw[->] (A) -- (B);
\draw[->] (B) -- (C);
\draw[->] (C) -- (A);
\end{tikzpicture}
```

5 Summary

5.1 Dummy section

Dummy text to test references:

See Knuth (1984) for additional discussion of literate programming.

References

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. <https://doi.org/10.1093/comjnl/27.2.97>.

Example watermark