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

The Authors

2024-11-16

This book is typeset with XeLaTeX (via Quarto and pandoc) in the Charis, Vazirmatn, and Amiri typefaces.

Table of contents

Preface

بسم الله الرحمن الرحيم

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).

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 code for this book can be used as a template for RTL document projects. 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:

```
[هذا نص عربي.]{.ar}
```

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

```
:::{.ar}
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.
:::
```

Rebuild the project with

```
quarto render
```

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.

```
[هذا نص عربي.]{.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">هذا نص عربي.</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:

```
[هذا نص عربي.]{.reg-ar-txt lang="ar"}
```

The output Latex code will be something like:

```
\foreignlanguage{arabic}{هذا نص 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 نَصٌّ عَرَبِيٌّ. 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"}
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.
:::
```

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

```
<div class="reg-ar-text" lang="ar" dir="rtl">
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.
</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'}
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.
:::
```

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}

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

\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 fiters

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
```

```
      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:

```
[هذا نص عربي.]{.ar}
```

Input for an Arabic div:

```
:::{.ar}
هذا نص عربي طويل.
سطرين. النص يبلغ حتى أكتب أن أريد
الخارجي. الملف لإنتاج قوارطو برنامج أستعمل
و هو برنامج يجي قد خلف البرنامج الذي بكداؤن كنت من أستعمله قبل.
:::
```

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:
      - srctex/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.

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	ʾ	E
ا	$\bar{a}$	A
ب	b	b
ت	t	t
ث	$\widehat{th}$	v
ج	j	j
ح	$\dot{h}$	H
خ	$\widehat{kh}$	x
د	d	d
ذ	$\widehat{dh}$	p
ر	r	r
ز	z	z
س	s	s
ش	$\widehat{sh}$	c
ص	$\dot{s}$	S
ض	$\dot{d}$	D
ط	$\dot{t}$	T
ظ	$\widehat{\dot{d}h}$	P
ع	ε	e
غ	$\widehat{gh}$	g
ف	f	f
ق	q	q

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

As you can see, I use digraphs (*d͡h*, *g͡h*, 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 inputing 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 *d͡hahabtu madh͡h·haban*.

Note the dot character · is automatically inserted by the filter between the digraph *d͡h* 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:

Ɛāʾiš͡hah and ʾĀdam are studying the Qurʾān and Ḥadīt͡h.

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 ʾ and `U+025b` for ɛ. 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

4 Summary

4.1 Test transliteration *abjd hwz ḥṭy klmn s fṣ qrsht thkhdh ḍghḍh*

Dummy

4.2 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.