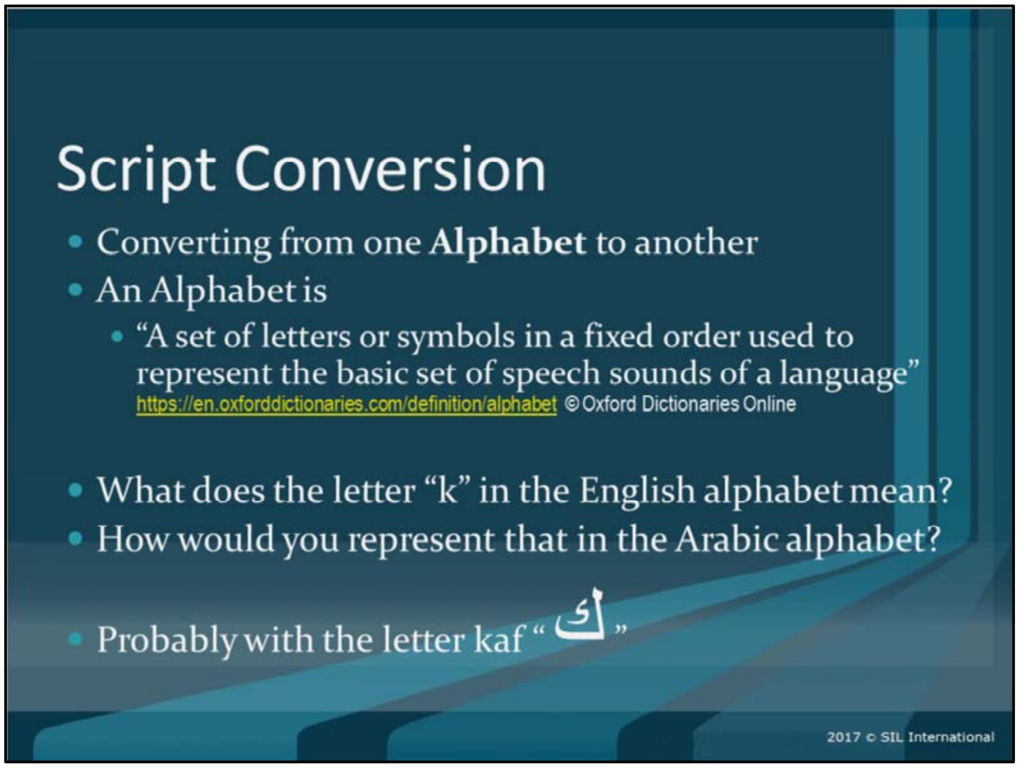
Script Conversion

By Jeff Heath

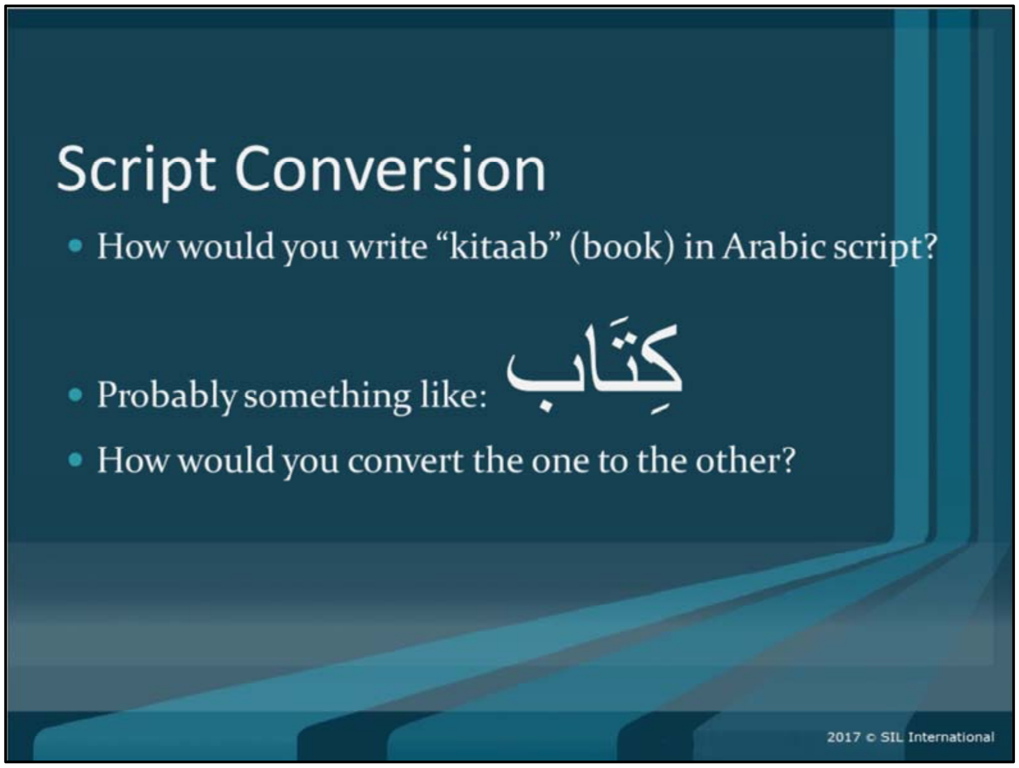
We’ll be talking in this session about script conversion.


Script Conversion

- Converting from one **Alphabet** to another
- An Alphabet is
  - "A set of letters or symbols in a fixed order used to represent the basic set of speech sounds of a language" https://en.oxforddictionaries.com/definition/alphabet © Oxford Dictionaries Online
- What does the letter "k" in the English alphabet mean?
- How would you represent that in the Arabic alphabet?
- Probably with the letter kaf " ك "

2017 © SIL International

Script Conversion is basically converting a text from one Alphabet to another.
Who can give me a definition of Alphabet?
□ Oxford says that an alphabet is…
□ So what does the letter "k" represent in the English alphabet? The sound [k]. Voiceless velar plosive.
□ How would you represent that in the Arabic alphabet?
□ Probably with the letter kaf.

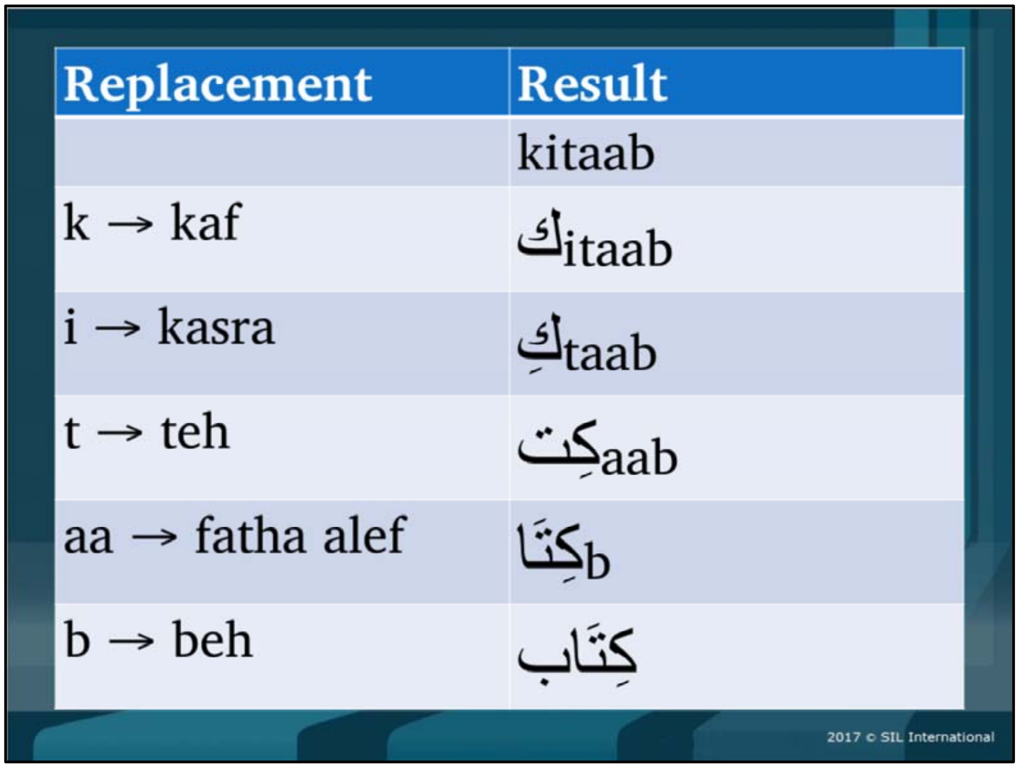


Script Conversion

- How would you write “kitaab” (book) in Arabic script?
- Probably something like: كِتَاب
- How would you convert the one to the other?

2017 © SIL International

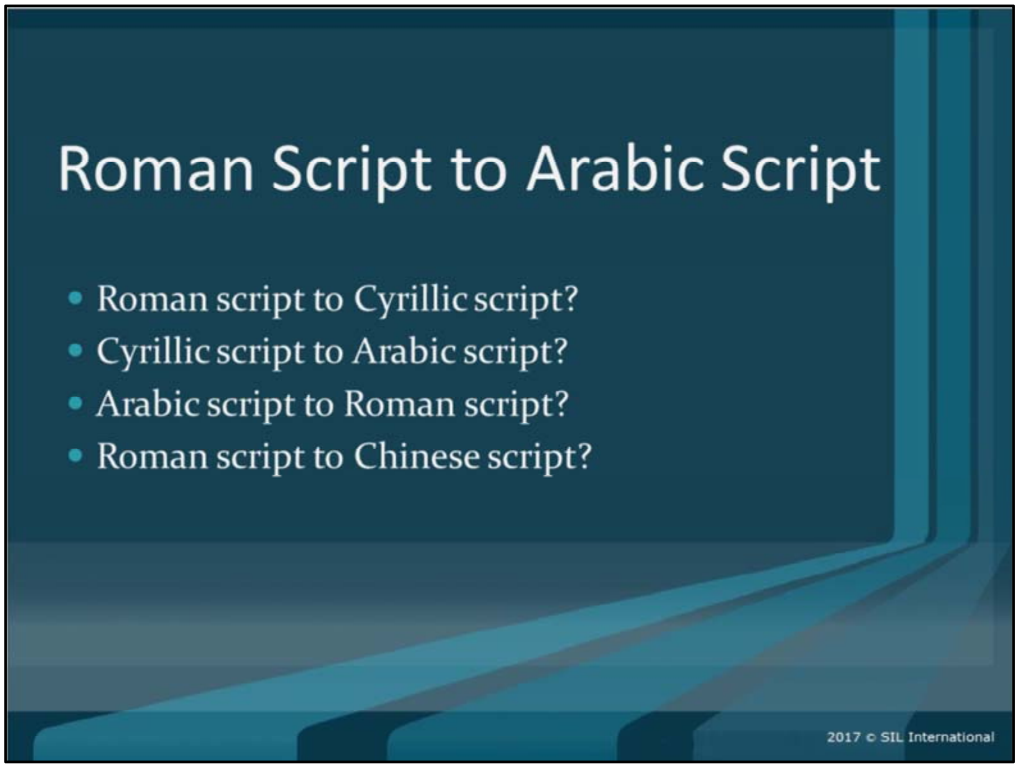
Let’s look at a little bit bigger chunk of text. How would you propose writing the word “kitaab” in Arabic script?
□ Here’s one possibility, if you’re fully vowelling the Arabic script text
□ How do you think you could convert one to the other?
Search/replace whole word
Disadvantage: You would have to know all of the possible words in the language!
Search/replace individual letters?
What about “a”? It converts to two different things!
Or rather the long “a” converts to something specific, if we can grab the two of them together.



Replacement	Result
	kitaab
k → kaf	كitaab
i → kasra	كِtaab
t → teh	كِتaab
aa → fatha alef	كِتَاb
b → beh	كِتَاب

2017 © SIL International

Let’s see what that looks like, if we replace individual letters, but make sure to treat the long “aa” as a unit.
What do you think? Does it look good? Yes!
I trust that everyone knows the Arabic letters take different shapes depending on whether they are word initial, word medial or word final, right? (So the TEH word final looks different than the one between two letters.)
I want you to note something here: What direction is the Roman script written? Left-to-right. What direction is the Arabic script written? Right-to-left. Did I do anything to make that change? No! Because we are using Unicode and smart fonts, as soon as we change to an Arabic letter the rendering engine knows that it is a strong right-to-left character and draws it in the string of characters as such.
It is fairly common that once you convert a text you will want to change the text direction of the entire paragraph, and we’ll see an example of that shortly.
Questions or comments?

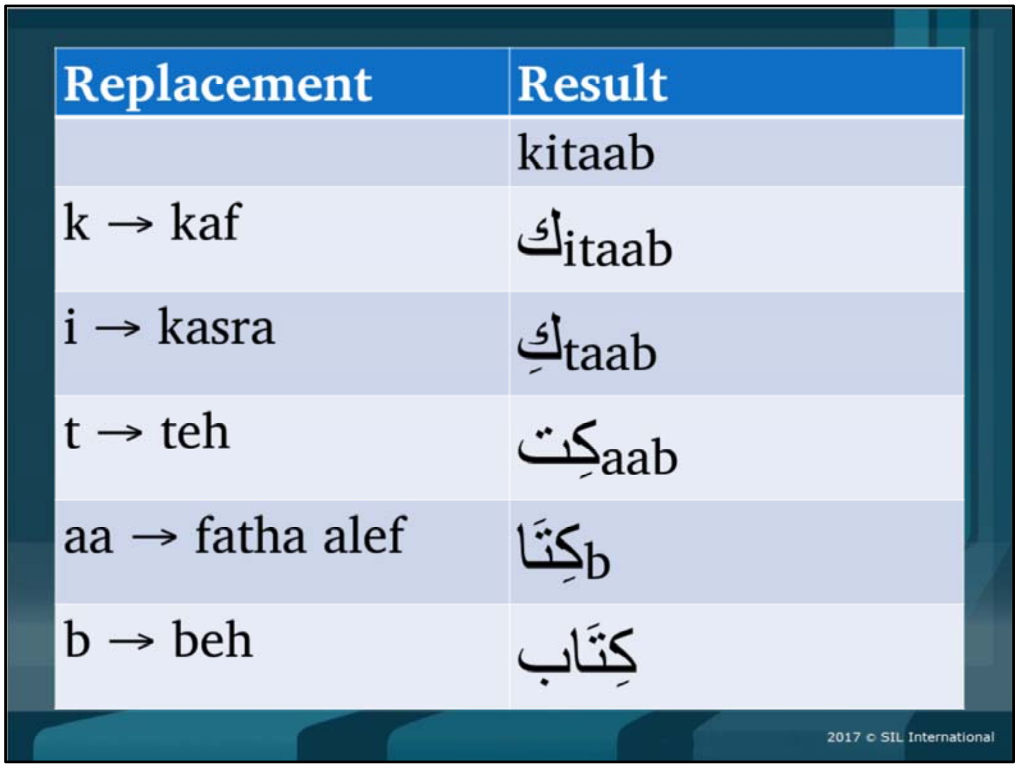


Roman Script to Arabic Script

- Roman script to Cyrillic script?
- Cyrillic script to Arabic script?
- Arabic script to Roman script?
- Roman script to Chinese script?

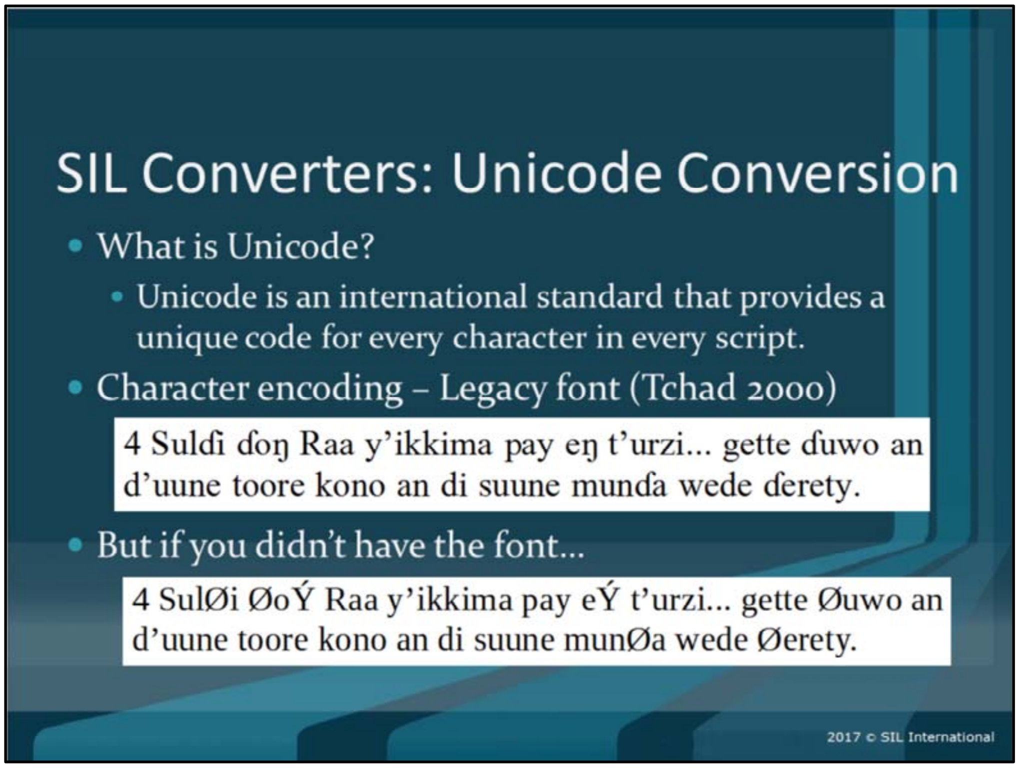
2017 © SIL International

So the example that we just did was basically a letter-to-letter Roman script to Arabic script conversion.
That’s the kind of conversion that we will mostly be focusing on.
But just to expand your minds a little bit…
□ Do you think you could use the same technique for Roman to Cyrillic script conversions? Yes
□ Cyrillic to Arabic? Yes
□ Arabic to Roman? Yes – you might have some different issues, but it’s still the same kind of conversion
□ Roman to Chinese? No – why not? Because it isn’t an alphabet in the same sense. We could still perform a conversion, but it would be more word-to-word instead of letter-to-letter.



Replacement	Result
	kitaab
k → kaf	كitaab
i → kasra	كِtaab
t → teh	كِتaab
aa → fatha alef	كِتَاb
b → beh	كِتَاب

OK so we've found a technique that seems to work fairly well for our conversion, but we don't want to have to do a bunch of manual search and replace operations whenever we want to do a script conversion. Fortunately we can take advantage of a tool called SIL Converters that has been created for a similar sort of operation.



SIL Converters: Unicode Conversion

- What is Unicode?
  - Unicode is an international standard that provides a unique code for every character in every script.
- Character encoding – Legacy font (Tchad 2000)

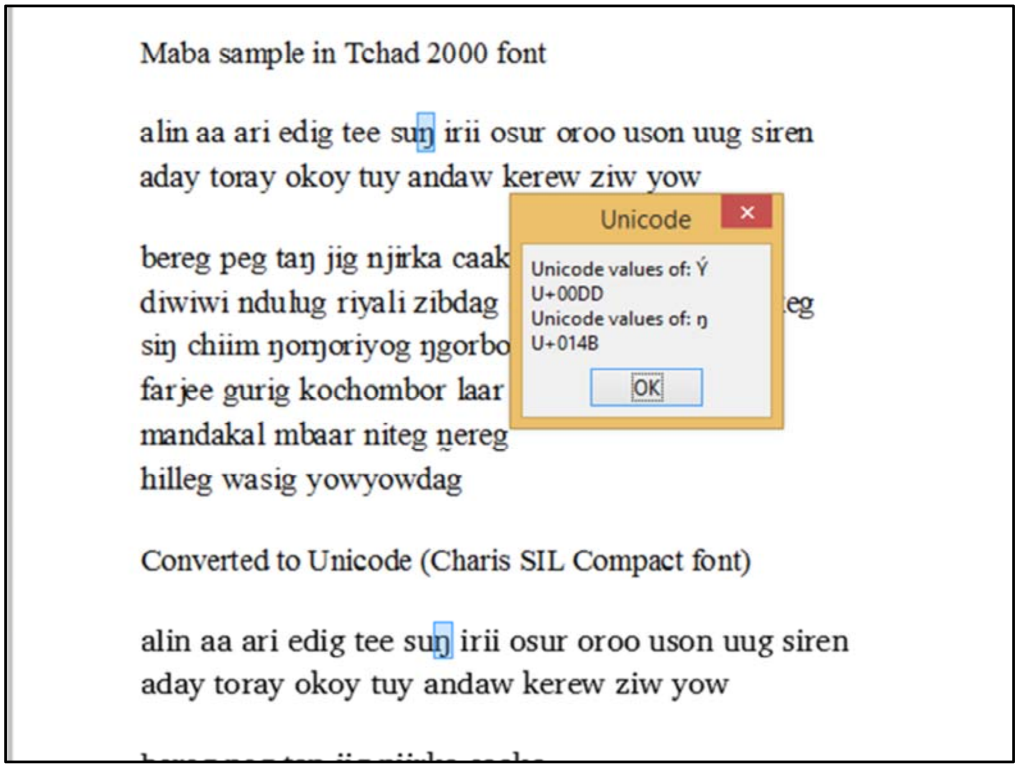
4 Suldi doŋ Raa y'ikkima pay eŋ t'urzi... gette duwo an d'uune toore kono an di suune munda wede derety.

- But if you didn't have the font...

4 SulØi ØoÝ Raa y'ikkima pay eÝ t'urzi... gette Øuwo an d'uune toore kono an di suune munØa wede Øerety.

2017 © SIL International

I believe SIL Converters was primarily created to perform Unicode conversions.
□ And since you're all Unicode experts now, who can give me a short definition of Unicode?
□ Unicode is an international standard that provides a unique code for every character in every script.
□ Back in the old days before Unicode, we had fonts that were limited to 240 characters or so, so we arranged characters as best we could to try to get access to all of the special characters that we needed for our region. These are called Legacy Fonts, and we created one in Chad called Tchad 2000, and we created a special keyboard to be able to type characters in that font.
□ Then we could use those to type text like this verse from the book of John in a language of Chad.
□ But if you send that text to someone who doesn't have that specific font, what happens?
□ You might end up with something like this. This is because the font they are using does not have the same encoding, or the same understanding of those 240 characters. Some of the characters are still the same, like a to z and punctuation. But many of the special characters like implosive d and eng have been garbled.
With Unicode, each character has a unique code, so when you use Unicode fonts, you can't have this kind of confusion any more, which is a very good thing!


Maba sample in Tchad 2000 font

alin aa ari edig tee suŋ irii osur oroo uson uug siren
aday toray okoy tuy andaw kerew ziw yow

bereg peg taŋ jig njirka caak
diwiwi ndulug riyali zibdag eg
siŋ chiim ŋoŋoriyog ŋgorbo
farjee gurig kochombor laar
mandakal mbaar niteg n̰ereg
hilleg wasig yowyowdag

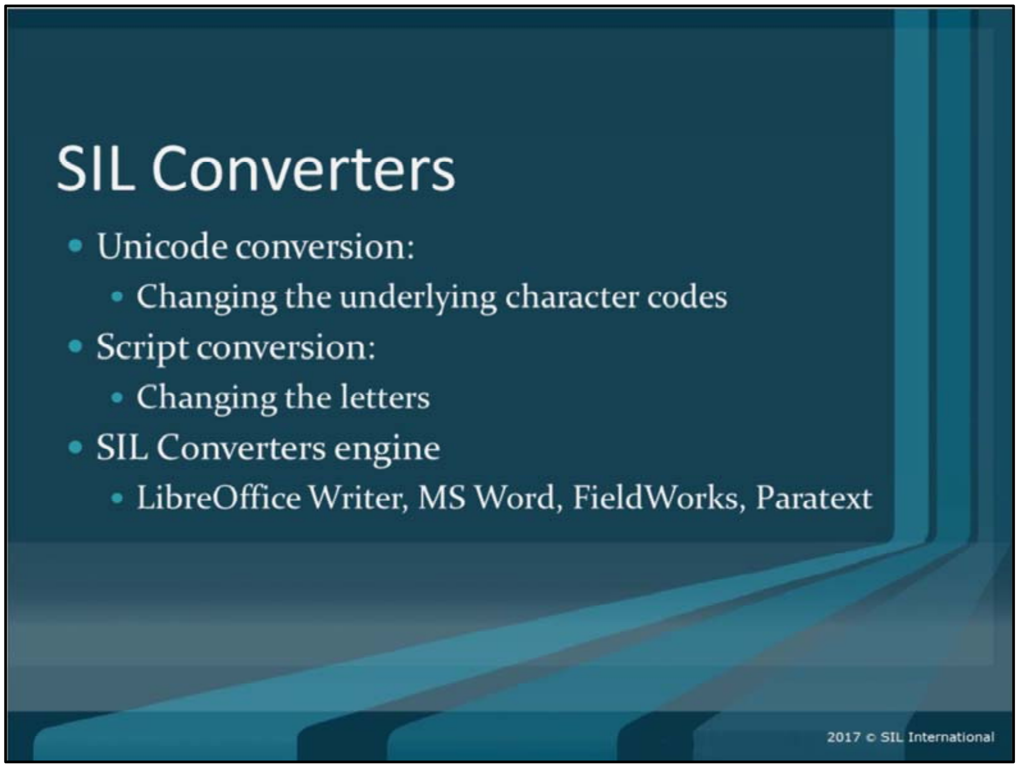
Unicode

Unicode values of: Ý
U+00DD
Unicode values of: ŋ
U+014B

Converted to Unicode (Charis SIL Compact font)

alin aa ari edig tee suŋ irii osur oroo uson uug siren
aday toray okoy tuy andaw kerew ziw yow

So if we have a document that was written with the Tchad 2000 font, like this text in the Maba language of Chad, we can run an SIL Converter on it that changes the Tchad 2000 encoding to the Unicode encoding.
That’s been done here, so under where it says “Converted to Unicode”, you see the same text as above, but now it is in the Charis SIL font, with Unicode encoding.
Two eng characters have been selected from the same position in these parallel texts, one in Tchad 2000, and one in Charis SIL. When a tool is run to show the underlying character codes, it shows that although the characters basically look the same, they have different underlying codes. The underlying code of the Tchad 2000 eng character is hexadecimal DD – but Unicode says that code means Y with acute accent. The code hexadecimal 14B is the correct Unicode code for the eng, and that’s the result after the conversion to Unicode.
Is anyone here not familiar with hexadecimal numbers? (Explain offline, basically numbers 0-9, letters A-F. For our purposes you can just think of them as a series of numbers and letters that you need to copy.)

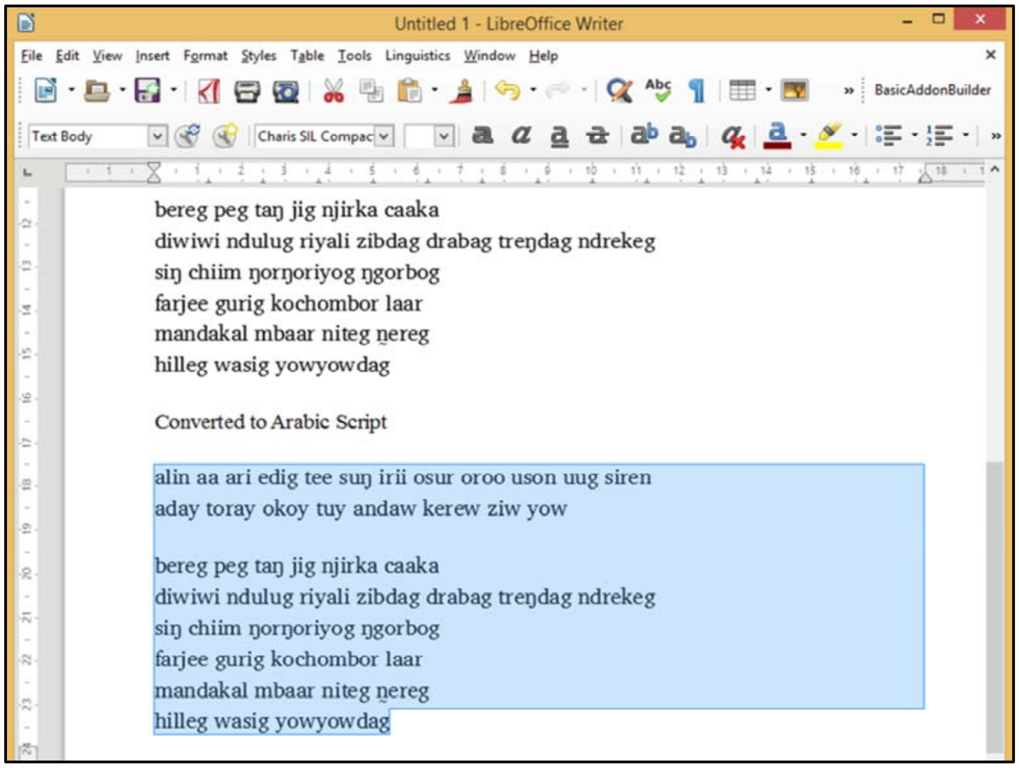


SIL Converters

- Unicode conversion:
  - Changing the underlying character codes
- Script conversion:
  - Changing the letters
- SIL Converters engine
  - LibreOffice Writer, MS Word, FieldWorks, Paratext

2017 © SIL International

So Unicode conversion is just the process of changing the underlying character codes of text, character-by-character, to conform with the Unicode standard.
□ And what we want is the conversion to Arabic script, letter-by-letter.
□ And it turns out that we can use the same SIL Converters tool to accomplish this task. SIL Converters really is more of a conversion engine, which can be used from a number of programs like the ones mentioned here.
But let’s jump right in with an example of converting Roman script to Arabic script in LibreOffice Writer.


Untitled 1 - LibreOffice Writer
File Edit View Insert Format Styles Table Tools Linguistics Window Help
BasicAddonBuilder
Text Body
Charis SIL Compac

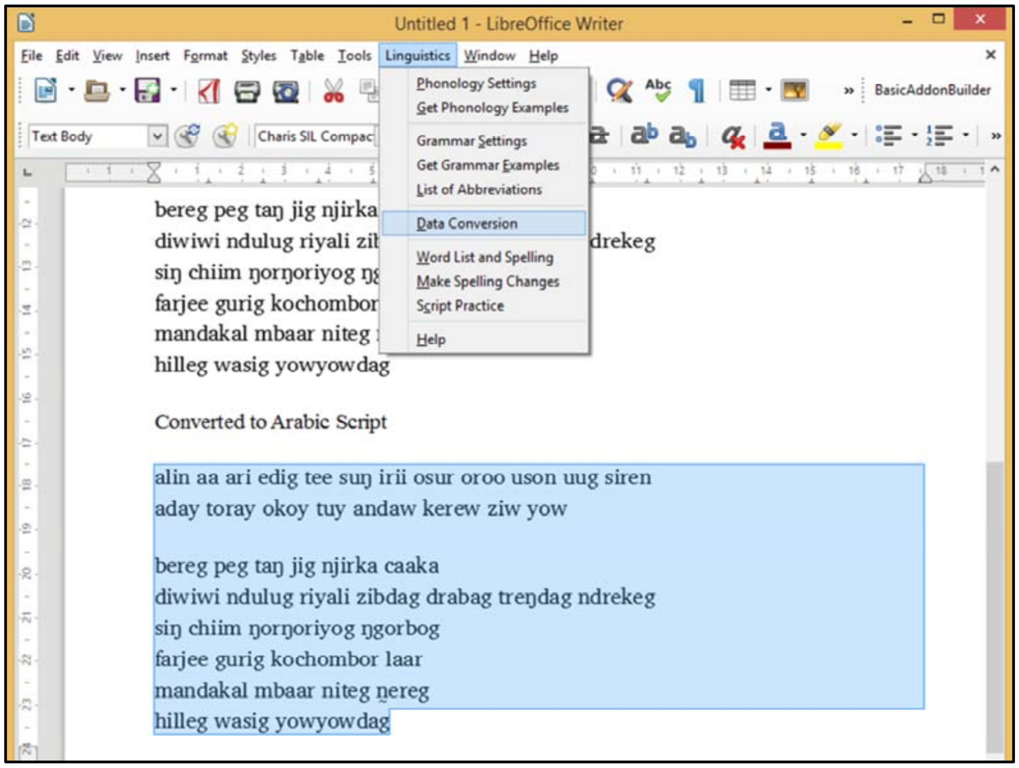
bereg peg taŋ jig njirka caaka
diwiwi ndulug riyali zibdag drabag treŋdag ndrekeg
siŋ chiim ŋorŋoriyog ŋgorbog
farjee gurig kochombor laar
mandakal mbaar niteg ṉereg
hilleg wasig yowyowdag

Converted to Arabic Script

alin aa ari edig tee suŋ irii osur oroo uson uug siren
aday toray okoy tuy andaw kerew ziw yow

bereg peg taŋ jig njirka caaka
diwiwi ndulug riyali zibdag drabag treŋdag ndrekeg
siŋ chiim ŋorŋoriyog ŋgorbog
farjee gurig kochombor laar
mandakal mbaar niteg ṉereg
hilleg wasig yowyowdag

Here we have our Maba text again, and we made a copy of the Unicode text, and pasted it under this title that says “Converted to Arabic Script”, and I’ve selected the text that I want to convert to Arabic script.
Note that I have a menu here named “Linguistics”. That’s because I installed the Linguistic Tools add-on.



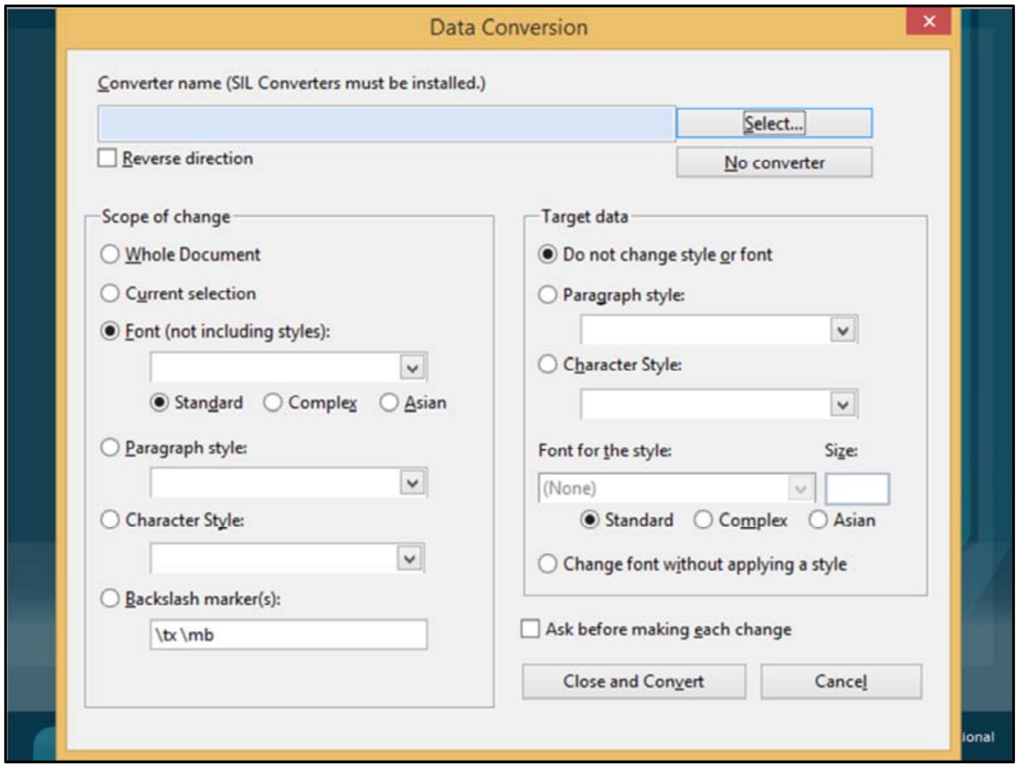
bereg peg taŋ jig njirka
diwiwi ndulug riyali zi… drekeg
siŋ chiim ŋorŋoriyog ŋ…
farjee gurig kochombor
mandakal mbaar niteg
hilleg wasig yowyowdag

Converted to Arabic Script

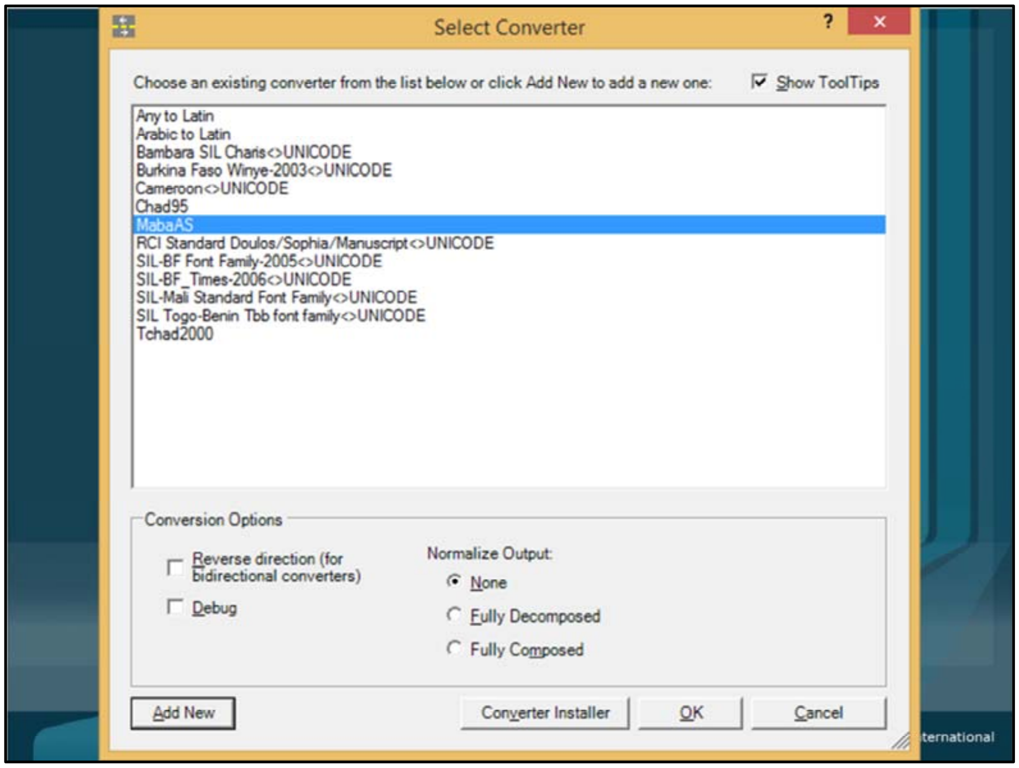
alin aa ari edig tee suŋ irii osur oroo uson uug siren
aday toray okoy tuy andaw kerew ziw yow

bereg peg taŋ jig njirka caaka
diwiwi ndulug riyali zibdag drabag treŋdag ndrekeg
siŋ chiim ŋorŋoriyog ŋgorbog
farjee gurig kochombor laar
mandakal mbaar niteg ŋereg
hilleg wasig yowyowdag

So I click on that Linguistics menu, and select Data Conversion



A window opens to ask me what kind of Data Conversion I want to do. The first field is the Converter name. I click on the Select button to the right.


Select Converter

Choose an existing converter from the list below or click Add New to add a new one:

[✓] Show ToolTips

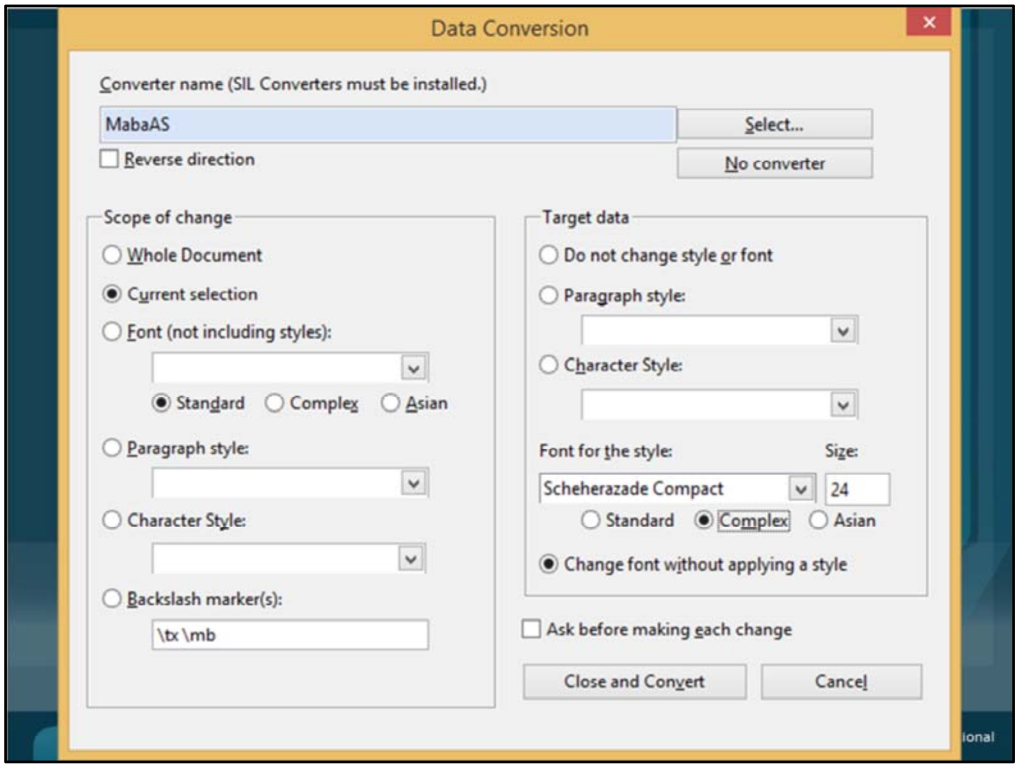
Any to Latin
Arabic to Latin
Bambara SIL Charis<>UNICODE
Burkina Faso Winye-2003<>UNICODE
Cameroon<>UNICODE
Chad95
MabaAS
RCI Standard Doulos/Sophia/Manuscript<>UNICODE
SIL-BF Font Family-2005<>UNICODE
SIL-BF_Times-2006<>UNICODE
SIL-Mali Standard Font Family<>UNICODE
SIL Togo-Benin Tbb font family<>UNICODE
Tchad2000

Conversion Options

[ ] Reverse direction (for bidirectional converters)
[ ] Debug

Normalize Output:
[✓ None]
Fully Decomposed
Fully Composed

In the list of available converters, I select MabaAS, then click OK.


Data Conversion

Converter name (SIL Converters must be installed.)

MabaAS

Select...

☐ Reverse direction

No converter

Scope of change
- ○ Whole Document
- ● Current selection
- ○ Font (not including styles):
  - ● Standard ○ Complex ○ Asian
- ○ Paragraph style:
- ○ Character Style:
- ○ Backslash marker(s): \tx \mb

Target data
- ○ Do not change style or font
- ○ Paragraph style:
- ○ Character Style:

Font for the style: Scheherazade Compact Size: 24

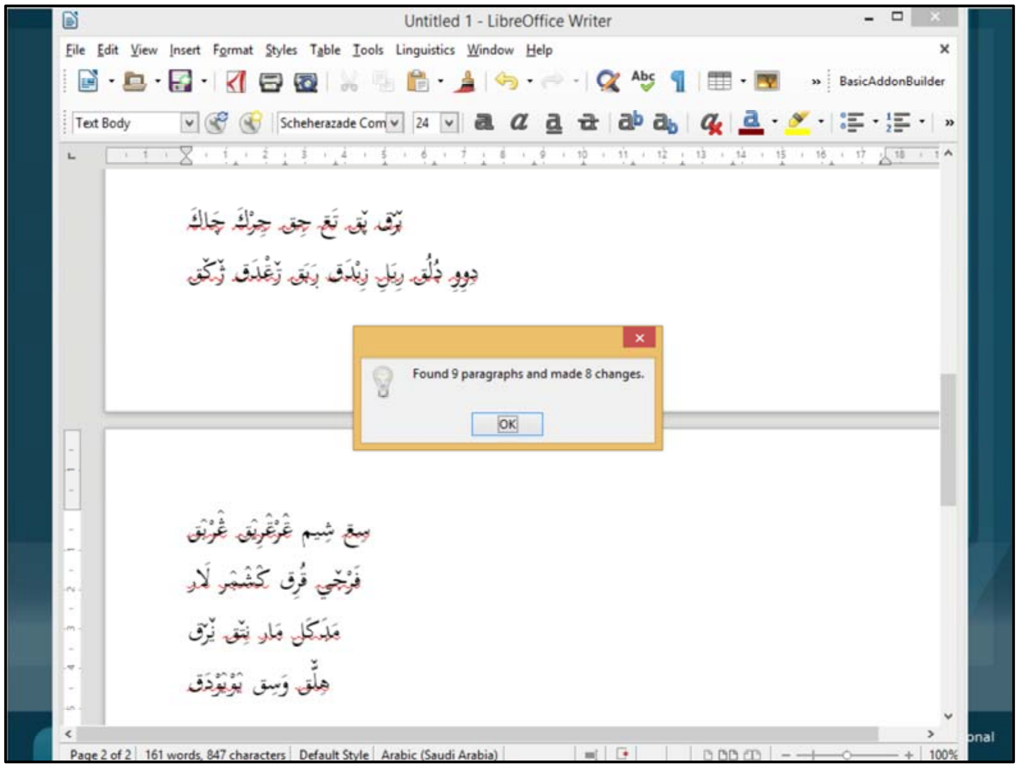
○ Standard ● Complex ○ Asian

● Change font without applying a style

☐ Ask before making each change

Close and Convert Cancel

Now the Converter name is filled in. Under Scope of change I select “Current selection”. And under Target data I tell it to Change to the Complex Scheherazade Compact font at 24 point without applying a style. Then I click on Close and Convert.



Untitled 1 - LibreOffice Writer

بَق بِق تَعَ حِق حِرْكَ حَالَكَ

دِوِو دُلُق رِيَلِ زِبْدَق رِبَق زَغْدَق ثَكَق

Found 9 paragraphs and made 8 changes.

OK

سِعَ شِيم غَرْغَرِبْق غُرْبَق

فَرْجِي قُرِق كَشْمَر لَار

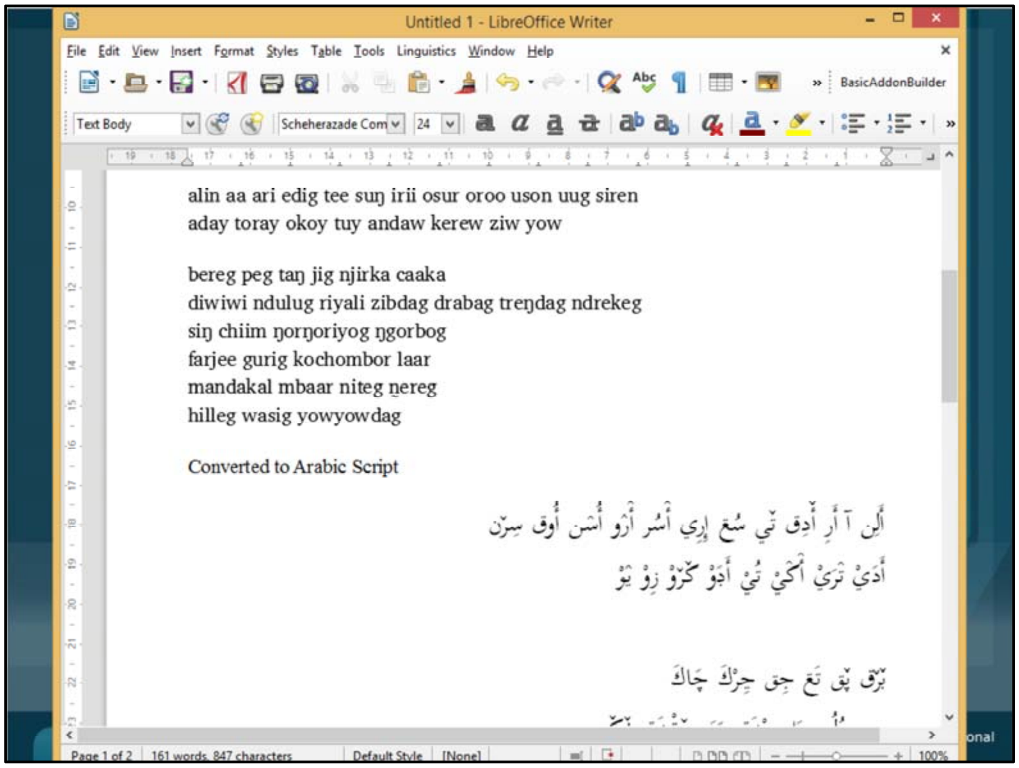
مَدَكَل مَار نِتَق نِرَق

هِلَّق وَسِق يَوْيَوْدَق

Page 2 of 2 | 161 words, 847 characters | Default Style | Arabic (Saudi Arabia) | 100%

File Edit View Insert Format Styles Table Tools Linguistics Window Help
BasicAddonBuilder
Text Body
Scheherazade Com
24

It performs the conversion and gives a summary; here it found 9 paragraphs and made 8 changes. I click OK.


Untitled 1 - LibreOffice Writer
File Edit View Insert Format Styles Table Tools Linguistics Window Help
BasicAddonBuilder
Text Body
Scheherazade Com
24
Page 1 of 2 | 161 words, 847 characters | Default Style | [None]
100%

alin aa ari edig tee suŋ irii osur oroo uson uug siren
aday toray okoy tuy andaw kerew ziw yow

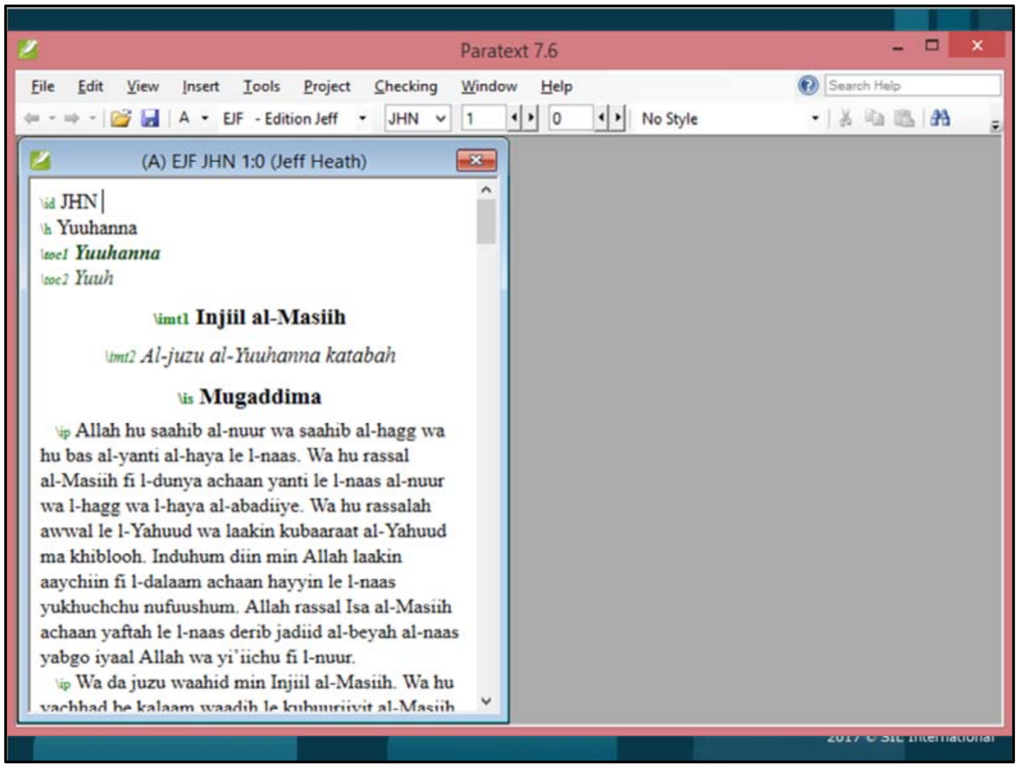
bereg peg taŋ jig njirka caaka
diwiwi ndulug riyali zibdag drabag treŋdag ndrekeg
siŋ chiim ŋorŋoriyog ŋgorbog
farjee gurig kochombor laar
mandakal mbaar niteg ṉereg
hilleg wasig yowyowdag

Converted to Arabic Script

أَلِن آ أَرِ أَدِق تّي سُّع إِرِي أُسُر أَرُو أُسُن أُوق سِرّن
أَدَيْ تَرَيْ أَكَيْ تُيْ أَدَوْ كَرّوْ زِوْ يَوْ

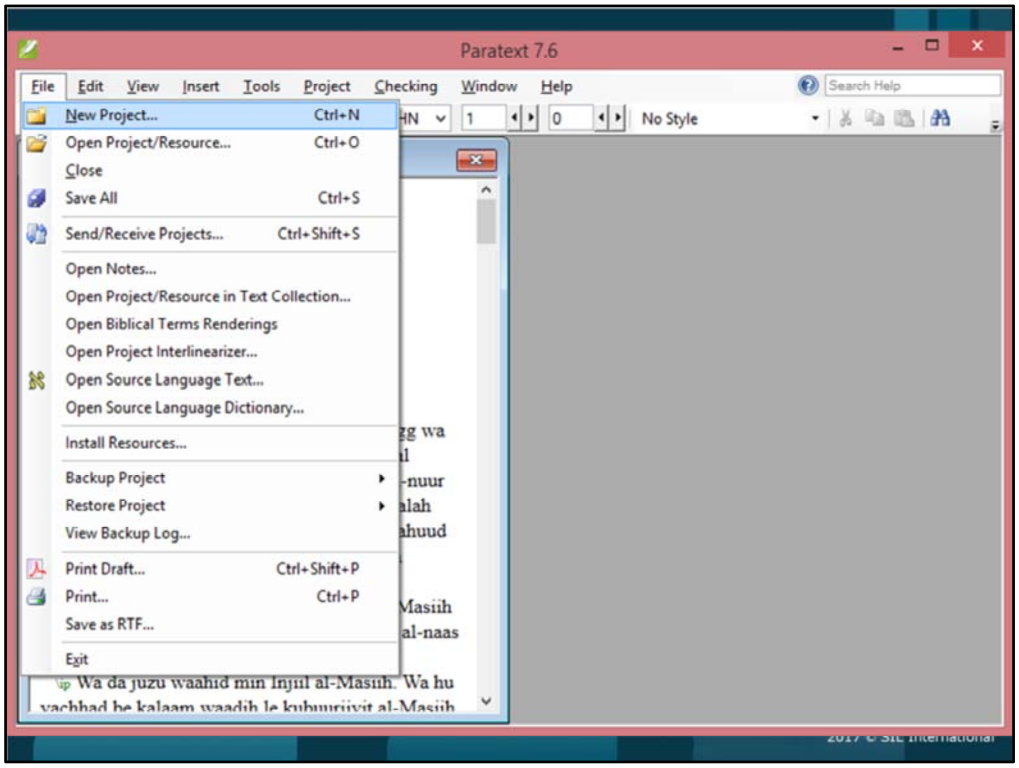
بّرّق پّق تَع جِق جِرْكَ چَاكَ

Then I select the Arabic script text that was produced, apply a Right-to-Left paragraph format, and I'm done!
Now that wasn't so hard, was it?
As you can imagine, there are some more difficult things that have to be done to get to this point, but once those harder one-time things are done – conversion to Arabic script can in fact be fairly easy.

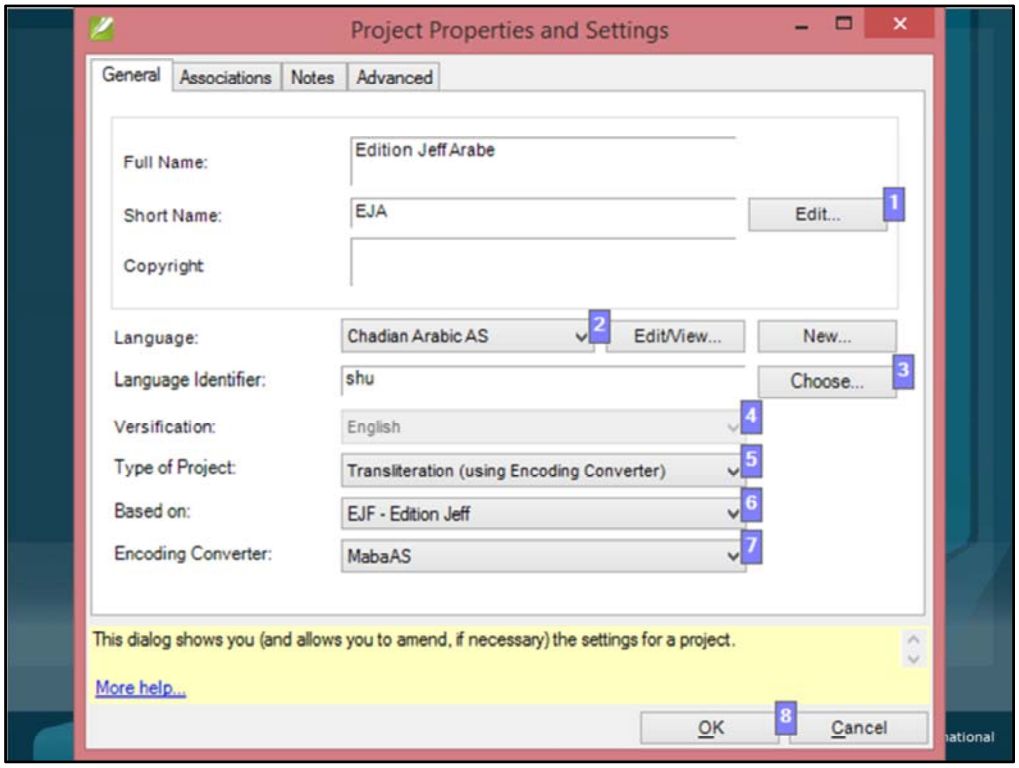

Paratext 7.6
File Edit View Insert Tools Project Checking Window Help
Search Help
A EJF - Edition Jeff JHN 1 0 No Style
(A) EJF JHN 1:0 (Jeff Heath)
2017 © SIL International

```
\id JHN
\h Yuuhanna
\toc1 Yuuhanna
\toc2 Yuuh
\imt1 Injiil al-Masiih
\imt2 Al-juzu al-Yuuhanna katabah
\is Mugaddima
\ip Allah hu saahib al-nuur wa saahib al-hagg wa
hu bas al-yanti al-haya le l-naas. Wa hu rassal
al-Masiih fi l-dunya achaan yanti le l-naas al-nuur
wa l-hagg wa l-haya al-abadiiye. Wa hu rassalah
awwal le l-Yahuud wa laakin kubaaraat al-Yahuud
ma khiblooh. Induhum diin min Allah laakin
aaychiin fi l-dalaam achaan hayyin le l-naas
yukhuchchu nufuushum. Allah rassal Isa al-Masiih
achaan yaftah le l-naas derib jadiid al-beyah al-naas
yabgo iyaal Allah wa yi'iichu fi l-nuur.
\ip Wa da juzu waahid min Injiil al-Masiih. Wa hu
yachhad be kalaam waadih le kubuuriiyit al-Masiih
```

Before we get into a more complete understanding of the difficult things, let's look at one more simple thing which is pretty cool! Here we have a Roman script Scripture project in Paratext 7.6 called Edition Jeff.


Paratext 7.6
File
Edit
View
Insert
Tools
Project
Checking
Window
Help
Search Help
New Project... Ctrl+N
Open Project/Resource... Ctrl+O
Close
Save All Ctrl+S
Send/Receive Projects... Ctrl+Shift+S
Open Notes...
Open Project/Resource in Text Collection...
Open Biblical Terms Renderings
Open Project Interlinearizer...
Open Source Language Text...
Open Source Language Dictionary...
Install Resources...
Backup Project
Restore Project
View Backup Log...
Print Draft... Ctrl+Shift+P
Print... Ctrl+P
Save as RTF...
Exit
No Style

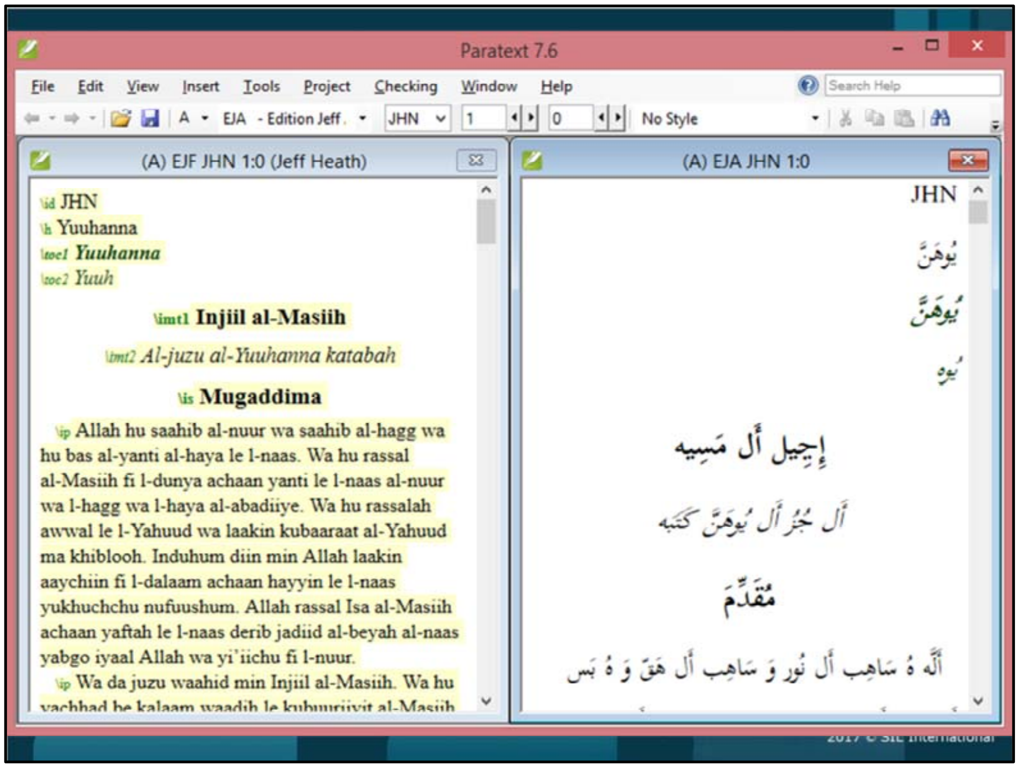
Let’s create a new project, selecting New Project from the file menu.


Project Properties and Settings

Full Name:	Edition Jeff Arabe
Short Name:	EJA
Copyright	
Language:	Chadian Arabic AS
Language Identifier:	shu
Versification:	English
Type of Project:	Transliteration (using Encoding Converter)
Based on:	EJF - Edition Jeff
Encoding Converter:	MabaAS

This dialog shows you (and allows you to amend, if necessary) the settings for a project.

Paratext asks us for the settings for this new project. Let’s call this project Edition Jeff Arabe, set up the language information, then select the Type of Project as “Transliteration (using Encoding Converter)”. Once you select that type of project, you have to select which project it is based on, and we select Edition Jeff, and which Encoding Converter to use, and we select MabaAS. When we click OK, what do you think we get?


Paratext 7.6
File Edit View Insert Tools Project Checking Window Help
Search Help
EJA - Edition Jeff. JHN 1 0 No Style

(A) EJF JHN 1:0 (Jeff Heath)

\id JHN
\h Yuuhanna
\toc1 *Yuuhanna*
\toc2 *Yuuh*

\imt1 **Injiil al-Masiih**

\imt2 *Al-juzu al-Yuuhanna katabah*

\is **Mugaddima**

\ip Allah hu saahib al-nuur wa saahib al-hagg wa hu bas al-yanti al-haya le l-naas. Wa hu rassal al-Masiih fi l-dunya achaan yanti le l-naas al-nuur wa l-hagg wa l-haya al-abadiiye. Wa hu rassalah awwal le l-Yahuud wa laakin kubaaraat al-Yahuud ma khiblooh. Induhum diin min Allah laakin aaychiin fi l-dalaam achaan hayyin le l-naas yukhuchchu nufuushum. Allah rassal Isa al-Masiih achaan yaftah le l-naas derib jadiid al-beyah al-naas yabgo iyaal Allah wa yi'iichu fi l-nuur.

\ip Wa da juzu waahid min Injiil al-Masiih. Wa hu

(A) EJA JHN 1:0

JHN

يُوهَنَّ

يُوهَنَّ

يُوه

إِجِيل أَل مَسِيه

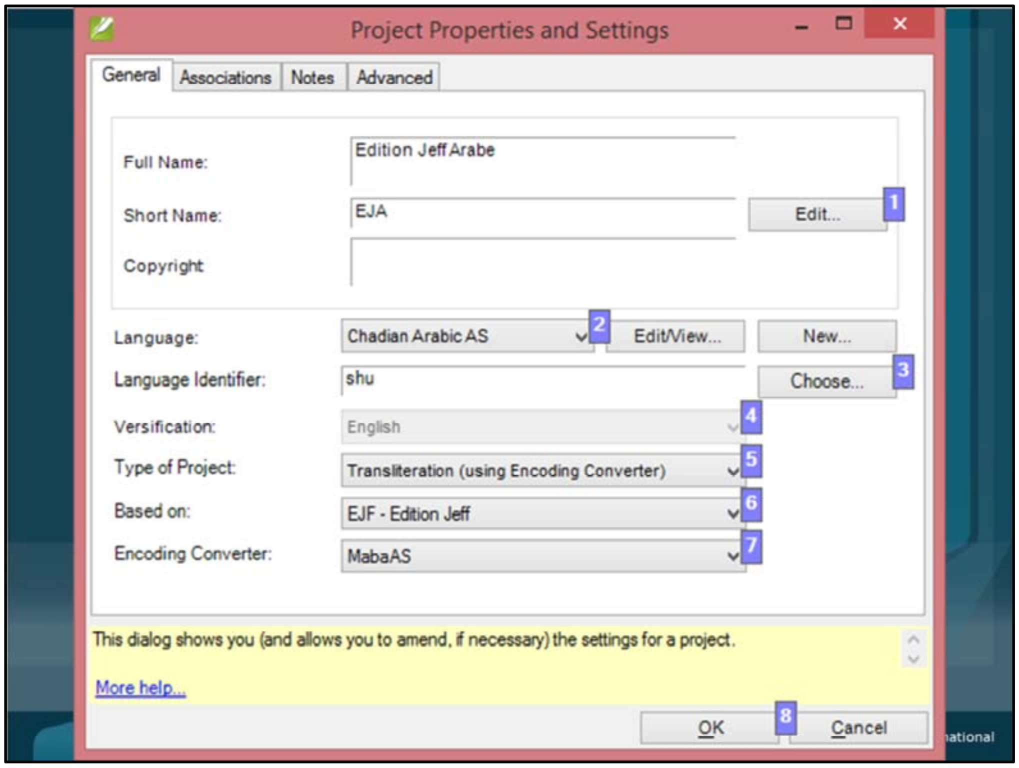
أَل جُزُ أَل يُوهَنَّ كَتَبه

مُقَدِّمَ

أَلَّه هُ سَاهِب أَل نُور وَ سَاهِب أَل هَقّ وَ هُ بَس

2017 © SIL International

We get a second Scripture project which is a complete conversion to Arabic script of the first! And if we make changes to the source Roman script translation, there is a menu item in the Arabic script project to refresh the transliteration, which will make sure they are in sync again.
Is that pretty cool, or what?!
Now obviously there had to be some magic in there somewhere. Can anyone tell me where the magic happened?


Project Properties and Settings

Full Name: Edition Jeff Arabe

Short Name: EJA

Copyright

Language: Chadian Arabic AS

Language Identifier: shu

Versification: English

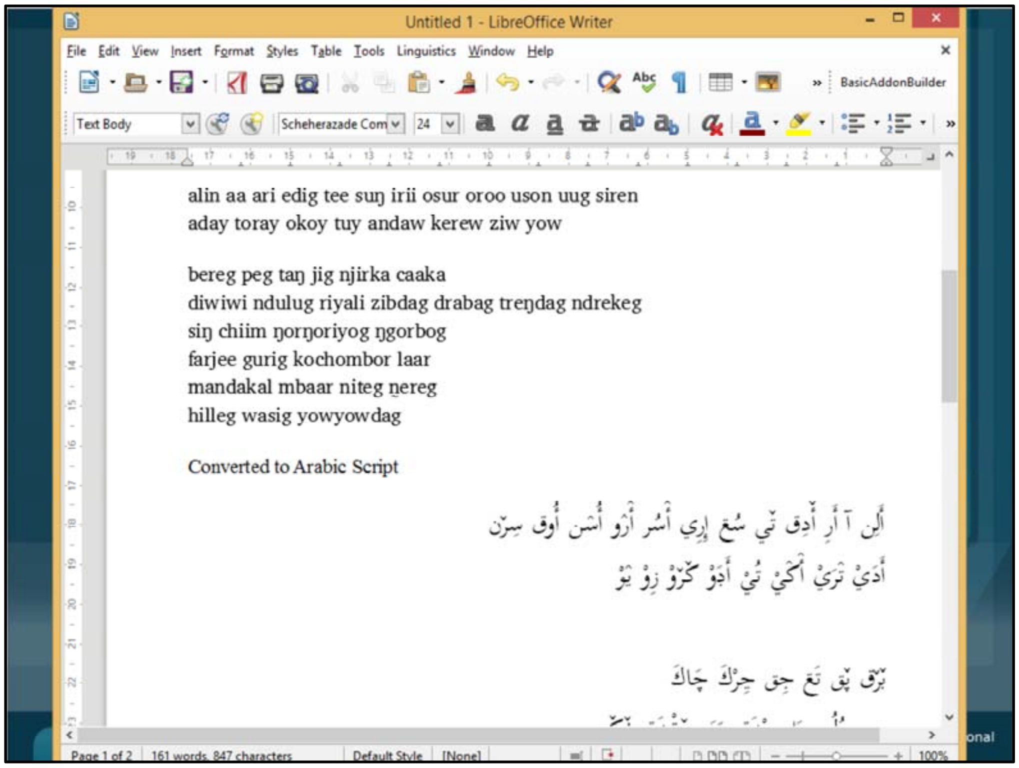
Type of Project: Transliteration (using Encoding Converter)

Based on: EJF - Edition Jeff

Encoding Converter: MabaAS

This dialog shows you (and allows you to amend, if necessary) the settings for a project.

It was in the selection of the Encoding Converter, in this case MabaAS. That's where all of the magic happens. And it's the making of that Encoding Converter – what we call a mapping file – that is the hard thing we have to do to make this magic happen. But the good news is that that hard part, making that mapping file, only has to happen once! And if we do a good job on it, it never needs to be touched again.


Untitled 1 - LibreOffice Writer

alin aa ari edig tee suŋ irii osur oroo uson uug siren
aday toray okoy tuy andaw kerew ziw yow

bereg peg taŋ jig njirka caaka
diwiwi ndulug riyali zibdag drabag treŋdag ndrekeg
siŋ chiim ŋorŋoriyog ŋgorbog
farjee gurig kochombor laar
mandakal mbaar niteg ɲereg
hilleg wasig yowyowdag

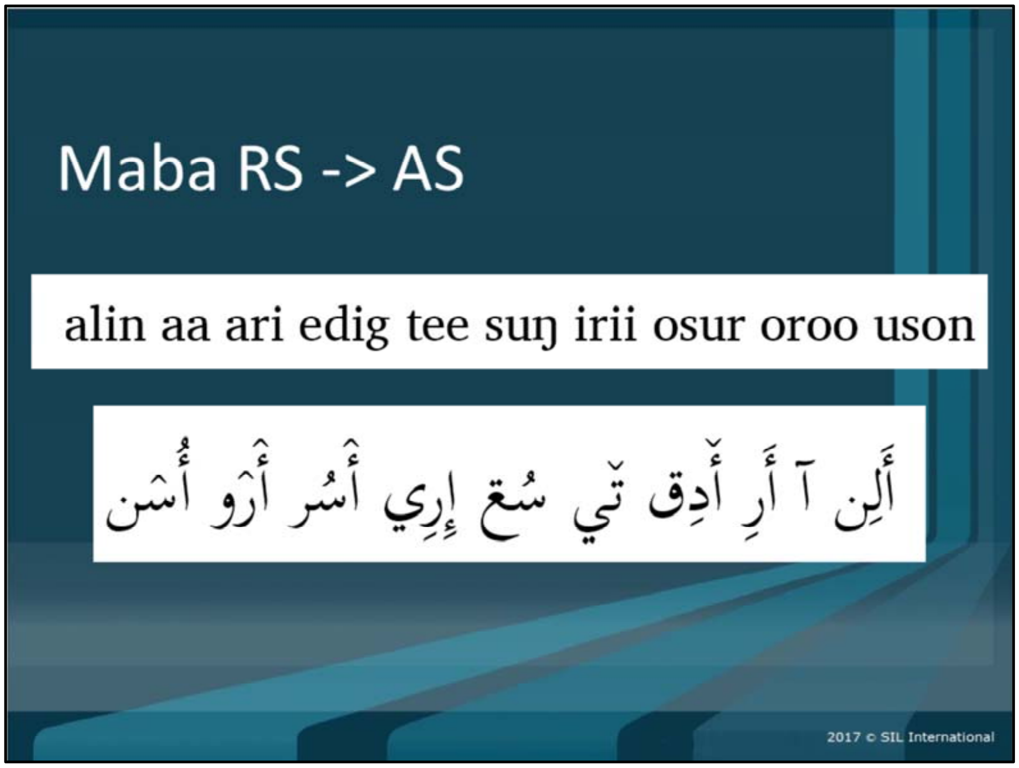
Converted to Arabic Script

ألِن آ أَرِ أٚدِق تٚي سُعٛ إِرِي أٛسُر أٛرُو أُشُن أُوق سِرٚن
أَدَيْ تٛرَيْ أٛكَيْ تُيْ أَدَوْ كٚرٚوْ زِوْ يٛوْ

بٚرٚق پٚق تَعٛ جِق جِرْكَ چَاكَ

Page 1 of 2 | 161 words, 847 characters | Default Style | [None] | 100%

To build up to understanding what we need to do to get that mapping file, let’s return to our Maba text that we converted. We’re going to take a closer look at part of the first line that was converted.


Maba RS -> AS

alin aa ari edig tee suŋ irii osur oroo uson

أَلِن آ أَرِ أ̌دِق ت̌ي سُڠ إِرِي أ̂سُر أ̂رو أُس̂ن

2017 © SIL International

So here is part of the first line in Roman script, and the output from the Arabic script conversion. To start with, I'd like you to make some observations about the conversion. What do you notice about the texts or the conversion itself?

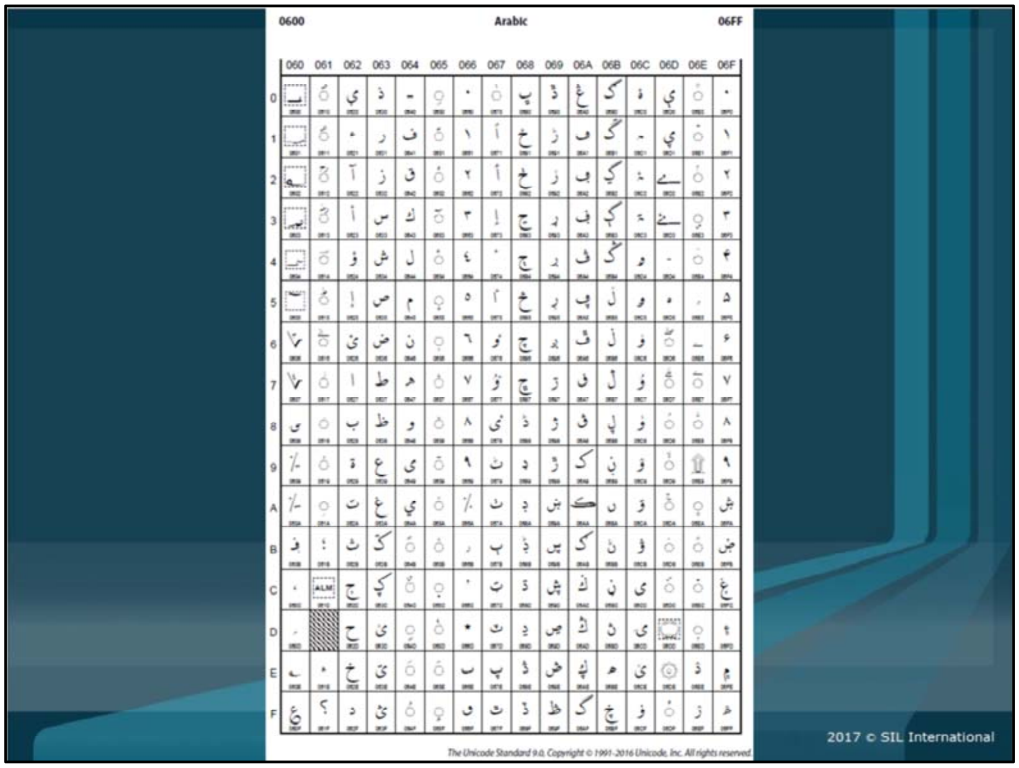
-I'll start you off: the RS text goes left to right, and the AS text goes right to left!

-Not standard Arabic, but most vowels/consonants use their normal MSA equivalents

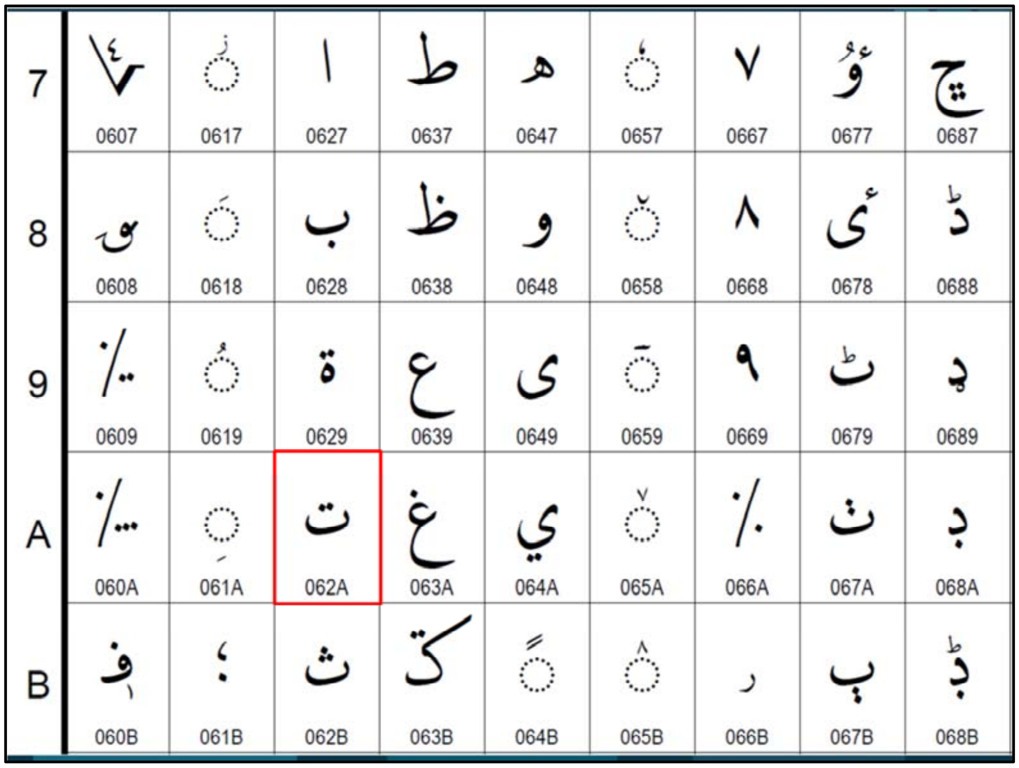
-E, o, ŋ turn into non-standard characters, but ones that are compatible with MSA (e.g. diacritics for vowels, eng is extension of ain character)

-Vowels at the beginning of a word need an alif hamza base to rest on

-Long vowels are made up of short vowel with waw or yah, except alef madda for long a (but that's word initial)

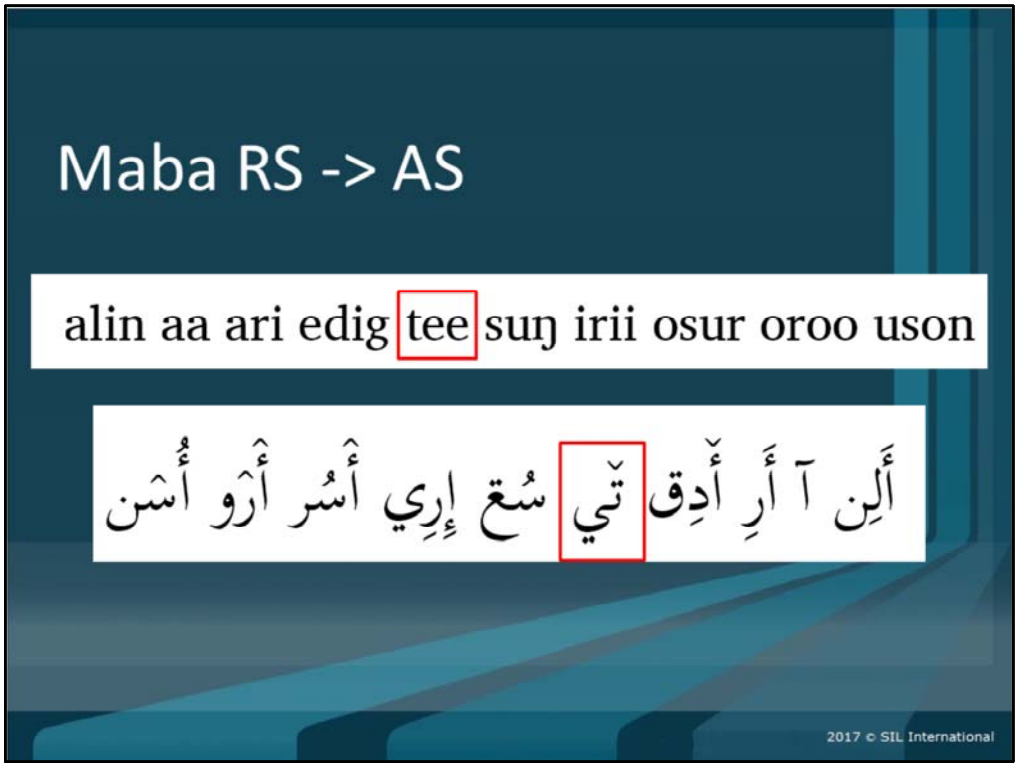

0600
Arabic
06FF
060 061 062 063 064 065 066 067 068 069 06A 06B 06C 06D 06E 06F
0 1 2 3 4 5 6 7 8 9 A B C D E F
The Unicode Standard 9.0. Copyright © 1991-2016 Unicode, Inc. All rights reserved.
2017 © SIL International

Who can tell me what this is?
This is a Unicode code chart. Specifically this is the main code chart for Arabic script. If you plan to do any script conversion, this is your friend! It starts at hexadecimal code 0600, so it's sometimes called the 600 code page.



7	؇ 0607	◌ؗ 0617	ا 0627	ط 0637	ه 0647	◌ٗ 0657	٧ 0667	ٷ 0677	ڇ 0687
8	؈ 0608	◌ؘ 0618	ب 0628	ظ 0638	و 0648	◌٘ 0658	٨ 0668	ٸ 0678	ڈ 0688
9	؉ 0609	◌ؙ 0619	ة 0629	ع 0639	ى 0649	◌ٙ 0659	٩ 0669	ٹ 0679	ډ 0689
A	؊ 060A	◌ؚ 061A	ت 062A	غ 063A	ي 064A	◌ٚ 065A	٪ 066A	ٺ 067A	ڊ 068A
B	؋ 060B	؛ 061B	ث 062B	ػ 063B	◌ً 064B	◌ٛ 065B	ٮ 066B	ٻ 067B	ڋ 068B

If we zoom in a little, you can see that the code chart shows various Arabic characters – some standard and some less so… – and it provides the unique Unicode code for each one. □ Let’s focus a bit on one character, the ARABIC LETTER TEH. We see here that this letter has the hexadecimal code 062A.



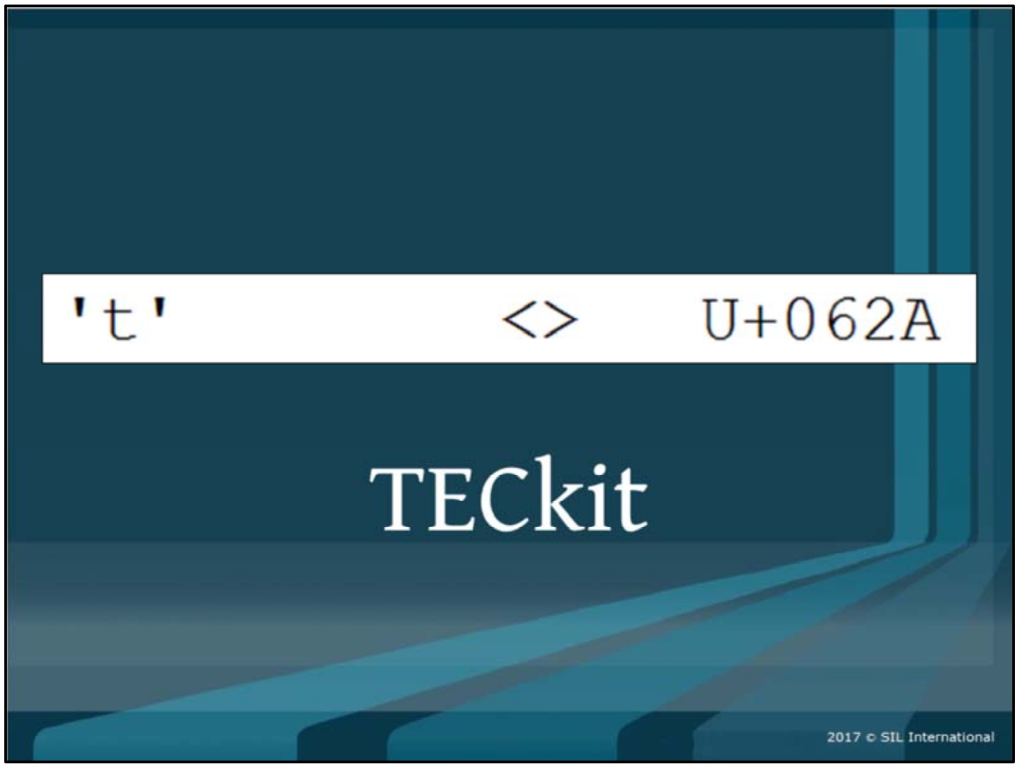
Maba RS -> AS

alin aa ari edig tee suŋ irii osur oroo uson

أَلِن آ أَرِ أ̌دِق ت̌ي سُغ إِرِي أ̂سُر أ̂رْو أُش̂ن

2017 © SIL International

Can anyone see where that letter TEH is used in the Arabic script line?
□ In this word here. (Fifth from the right. Remember that Arabic letters change shape depending on their position in the word!)
That corresponds to which word in the Roman script?
□ “tee”.
So what would you say is the Roman script equivalent for the Arabic letter TEH? It’s the letter “t”.
This leads us to the following…

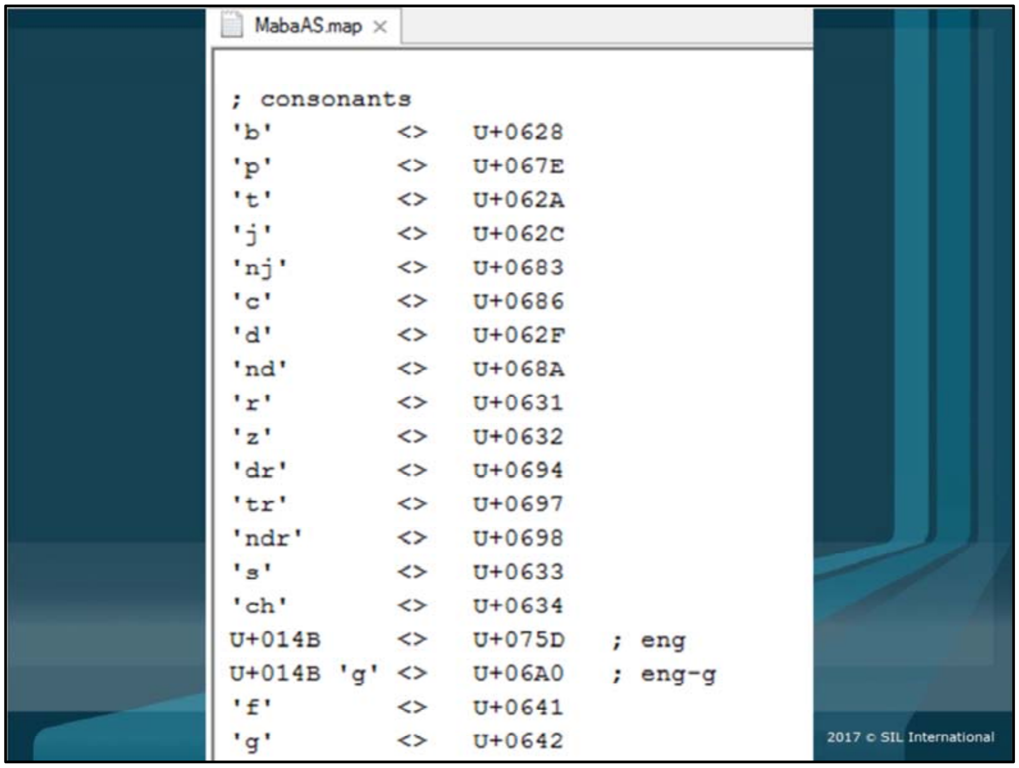


```
't'          <>    U+062A
```

TECkit

2017 © SIL International

…what do you think this is, and what does it mean?
It is one line from our conversion mapping file, and it means that the letter ‘t’ gets converted to the ARABIC LETTER TEH, which as we saw is Unicode code 062A.
It is part of the mapping file that accomplishes the conversion of Roman script to Arabic script in the Maba language.
□ This file is written in the TECkit language, which you could say is a kind of programming language, but before you get too nervous, the TECkit language isn’t nearly as complicated as a lot of programming languages. Every one of you here could create a mapping file if you set your mind to it, even if you have avoided computer programming before!
And this one line is our first lesson in the TECkit language.
First we see that we can specify a Roman script character by putting it between quotes. And we can specify a Unicode character by just putting a “U+” in front of the 4-digit character code that you can find on the code page.
Next we look at the less-than and greater-than signs. These signs are used to indicate what kind of conversions are allowed. The greater-than sign to the right means that the converter can convert the ‘t’ to the Unicode character 062A. The less-then sign to the left means that the converter can convert that TEH back to the letter ‘t’, if you ask it to reverse the direction of the conversion. We’ll talk about that more later, but for the moment we just say that these two letters are equivalent between the two scripts and we use both the less-than and greater-than signs to indicate that we can convert back and forth between them.



MabaAS.map

```
; consonants
'b'         <>   U+0628
'p'         <>   U+067E
't'         <>   U+062A
'j'         <>   U+062C
'nj'        <>   U+0683
'c'         <>   U+0686
'd'         <>   U+062F
'nd'        <>   U+068A
'r'         <>   U+0631
'z'         <>   U+0632
'dr'        <>   U+0694
'tr'        <>   U+0697
'ndr'       <>   U+0698
's'         <>   U+0633
'ch'        <>   U+0634
U+014B      <>   U+075D   ; eng
U+014B 'g'  <>   U+06A0   ; eng-g
'f'         <>   U+0641
'g'         <>   U+0642
```

2017 © SIL International

Now let me show you a little bit more of our TECkit mapping file. This is part of the MabaAS.map file, for converting Roman script to Arabic script in the Maba language. On the fourth line down we see our ‘t’ to TEH conversion line, and there are a number of lines that are similar to it, showing the conversion from something in Roman script to a Unicode character on the Arabic 600 code page. What else do you notice here, or what questions do you have?

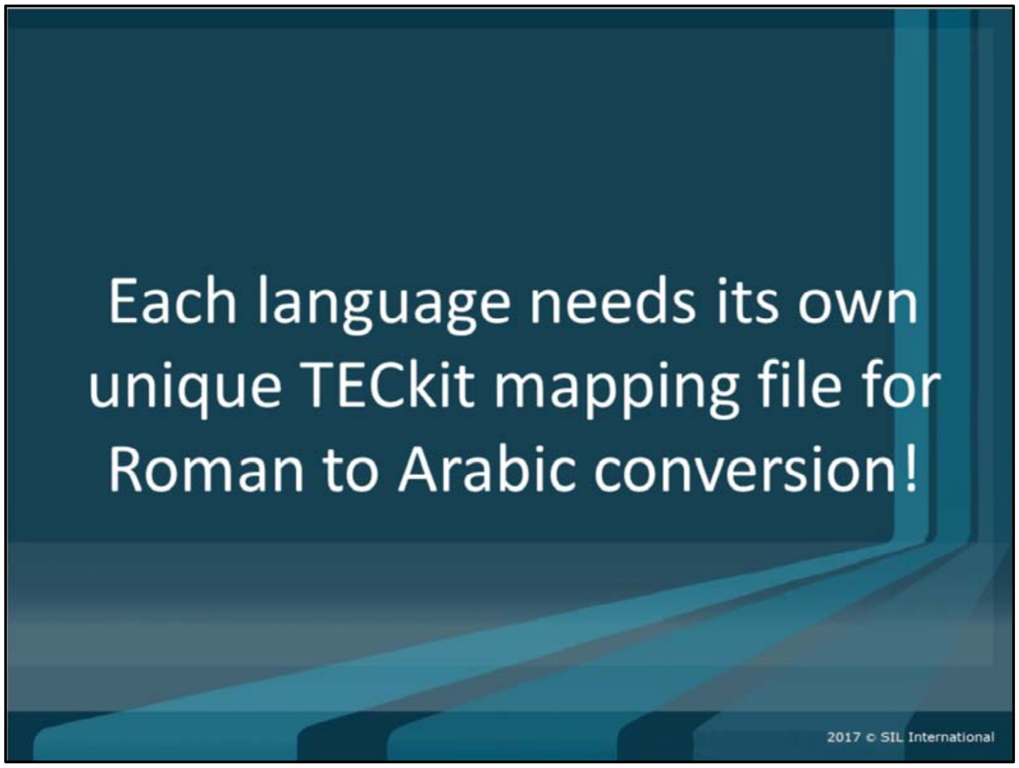
Semi-colons: mark comments that don’t affect the conversion, like the comment at the top that indicates that this is the section of the conversion mapping file that deals with consonants. Everything on the line after the semi-colon is a comment.

Multi-character matches: It may not be just a single character that we need to convert, but a digraph, or an even longer string. The rules are always tested from the most to least specific, so a longer string would be matched over a shorter string. So, for example, if a “t” is followed by an “r”, the two characters together will get converted to a Unicode 0697 – they won’t get converted individually to a TEH and REH.

Unicode on left: For the lines where we want to convert the eng character, rather than typing the eng between quotes, I chose to put the Unicode character code in, and then added a comment at the end of the line to remind me that those are eng characters.

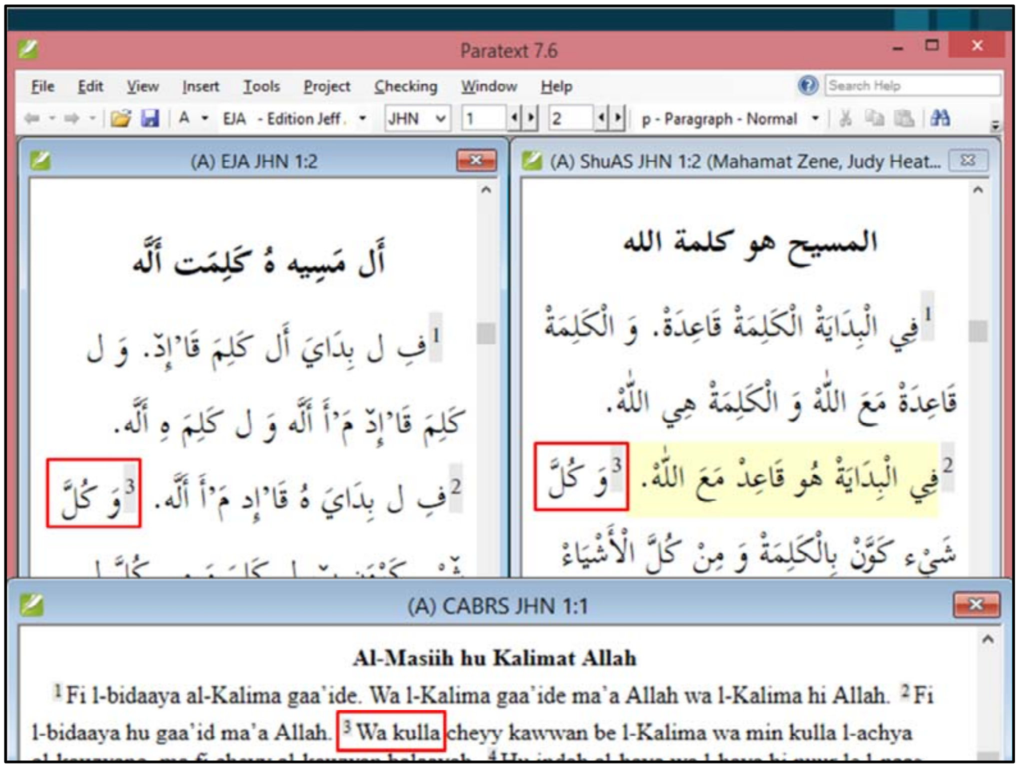
700 page: You may have also noticed that the Arabic script code for the eng is on the 700 code page. There is a partial page there called Arabic Supplement, which provides some additional Arabic script codes that couldn’t fit on the main 600 code page. Depending on the special characters you need in your language, this page may also be your friend!

Question for you: Would the mapping file for your language be the same? Would it be similar? Why or why not?



Each language needs its own unique TECkit mapping file for Roman to Arabic conversion!

And just in case that wasn't clear, let me say it again. "Each language needs its own unique TECkit mapping file for Roman to Arabic conversion!"


Paratext 7.6

File Edit View Insert Tools Project Checking Window Help

EJA - Edition Jeff. JHN 1 2 p - Paragraph - Normal

(A) EJA JHN 1:2

أَل مَسِيه ةُ كَلِمَت أَلّه

1 فِ ل بِدَايَ أَل كَلِمَ قَا'إِدّ. وَ ل كَلِمَ قَا'إِدّ مَ'أَ أَلّه وَ ل كَلِمَ هِ أَلّه. 2 فِ ل بِدَايَ ةُ قَا'إِد مَ'أَ أَلّه. 3 وَ كُلّ

(A) ShuAS JHN 1:2 (Mahamat Zene, Judy Heat...

المسيح هو كلمة الله

1 فِي الْبِدَايَةْ الْكَلِمَةْ قَاعِدَةْ. وَ الْكَلِمَةْ قَاعِدَةْ مَعَ اللّٰهْ وَ الْكَلِمَةْ هِي اللّٰهْ. 2 فِي الْبِدَايَةْ هُو قَاعِدْ مَعَ اللّٰهْ. 3 وَ كُلّ شَيْء كَوَّنْ بِالْكَلِمَةْ وَ مِنْ كُلّ الْأَشْيَاءْ

(A) CABRS JHN 1:1

Al-Masiih hu Kalimat Allah

1 Fi l-bidaaya al-Kalima gaa'ide. Wa l-Kalima gaa'ide ma'a Allah wa l-Kalima hi Allah. 2 Fi l-bidaaya hu gaa'id ma'a Allah. 3 Wa kulla cheyy kawwan be l-Kalima wa min kulla l-achya

As an example of that, do you remember that nifty AS conversion that I showed you in Paratext before, I used the Edition Jeff project as the base, which happened to be just some selections from the Chadian Arabic translation, like this Roman script at the bottom. On the left here I'm showing you the Edition Jeff Arabe that we produced. Do you recall the name of the mapping file that we used? MabaAS! Hmmm… what language do you think that was created for? Maba! OK, now on the right I've put the REAL Chadian Arabic AS text.

□ If we look first at the beginning of verse 3 – it's the same! But if we look pretty much anywhere else…

□ …the texts are different.

What do you notice as differences?

Articles aren't attached

Section heading is not vowelled

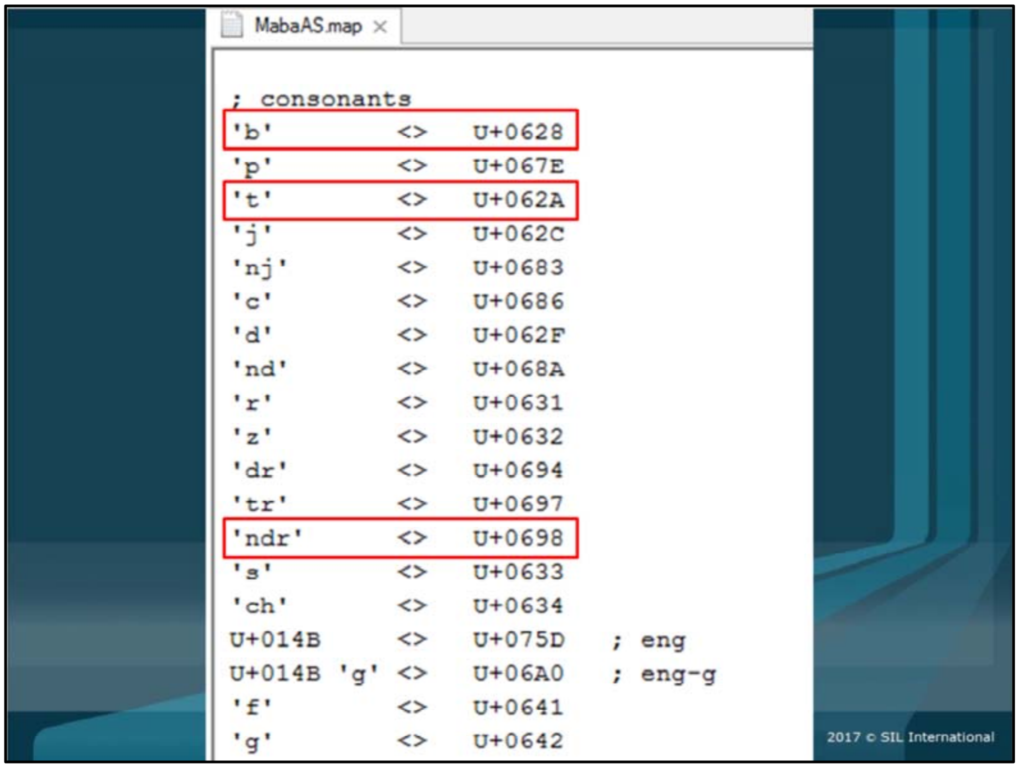
Allah

Vowels at the end of words are different

Use of some "special" vowels which aren't used in Chadian Arabic

Glottal not handled

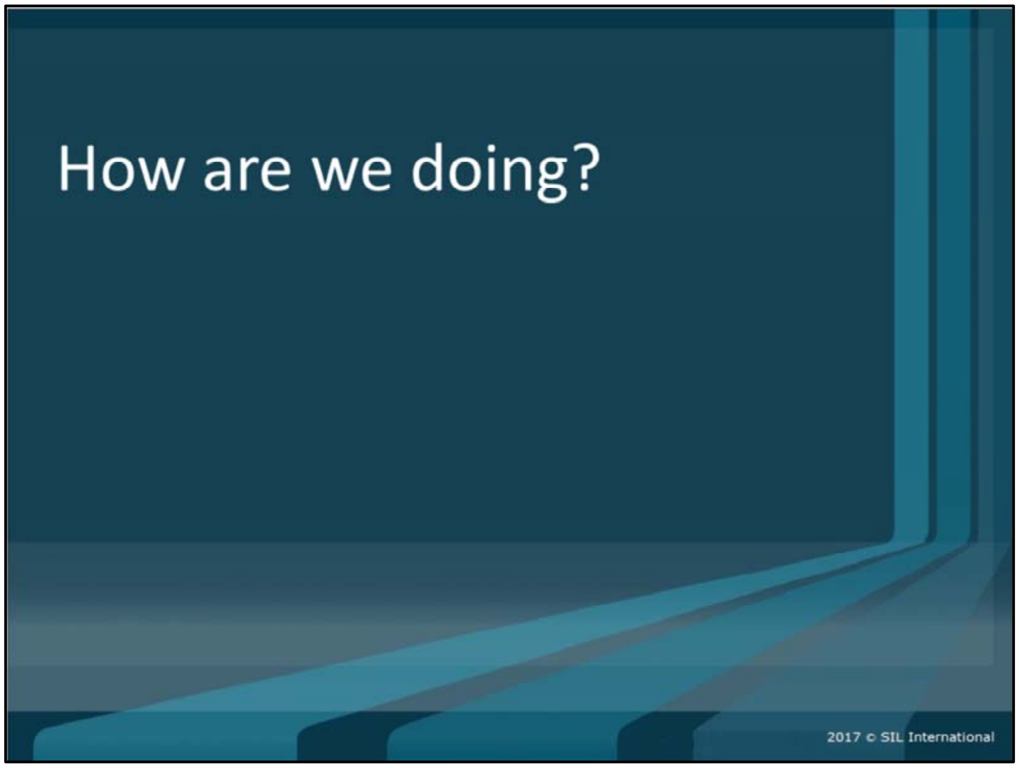
So what's the take-away? Each language needs to have its own TECkit mapping file for conversion to AS.


MabaAS.map

```
; consonants
'b'         <>   U+0628
'p'         <>   U+067E
't'         <>   U+062A
'j'         <>   U+062C
'nj'        <>   U+0683
'c'         <>   U+0686
'd'         <>   U+062F
'nd'        <>   U+068A
'r'         <>   U+0631
'z'         <>   U+0632
'dr'        <>   U+0694
'tr'        <>   U+0697
'ndr'       <>   U+0698
's'         <>   U+0633
'ch'        <>   U+0634
U+014B      <>   U+075D   ; eng
U+014B 'g'  <>   U+06A0   ; eng-g
'f'         <>   U+0641
'g'         <>   U+0642
```

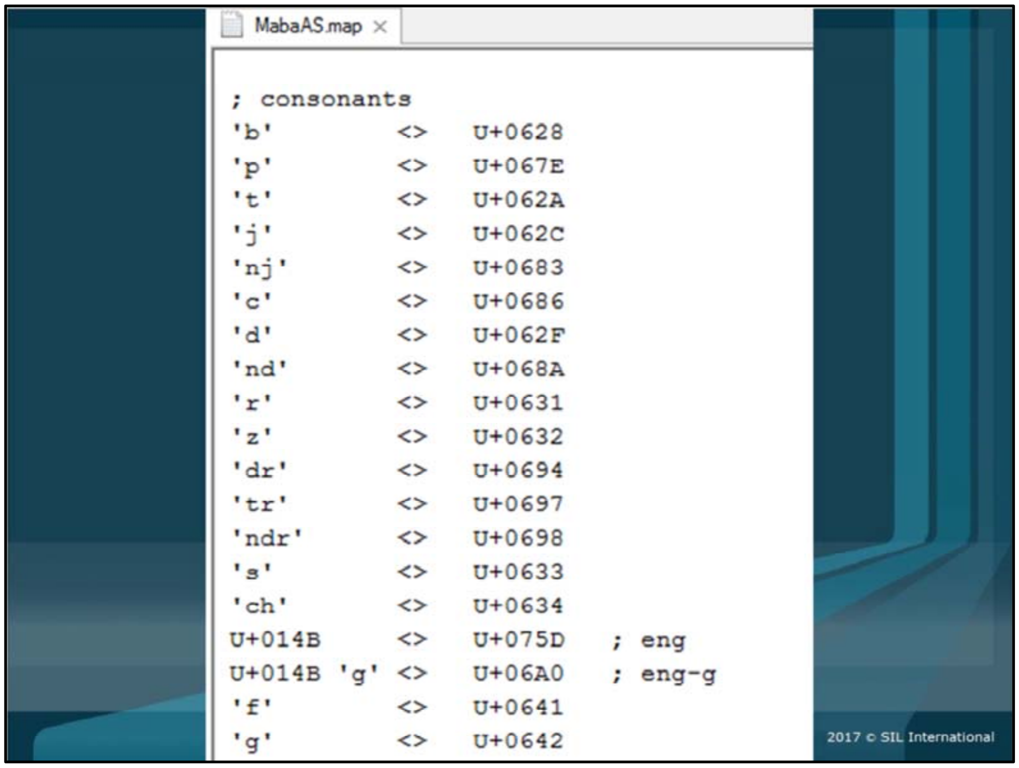
2017 © SIL International

But I also want you to hear what I'm **NOT** saying! I'm **NOT** saying that each TECkit mapping file has to be completely built from scratch.
□ You probably don't have an "ndr" trigraph in your language…
□ but you probably have a "b" and it probably converts to a "BEH".
□ You probably also have a "t" and it probably converts to a "TEH".
□ So you can borrow lines or sections or ideas from other mapping files to build up the mapping file that works for your language. And that's what we'll be doing this week for each of you that wants to create a mapping file for converting Roman script to Arabic script in your language.



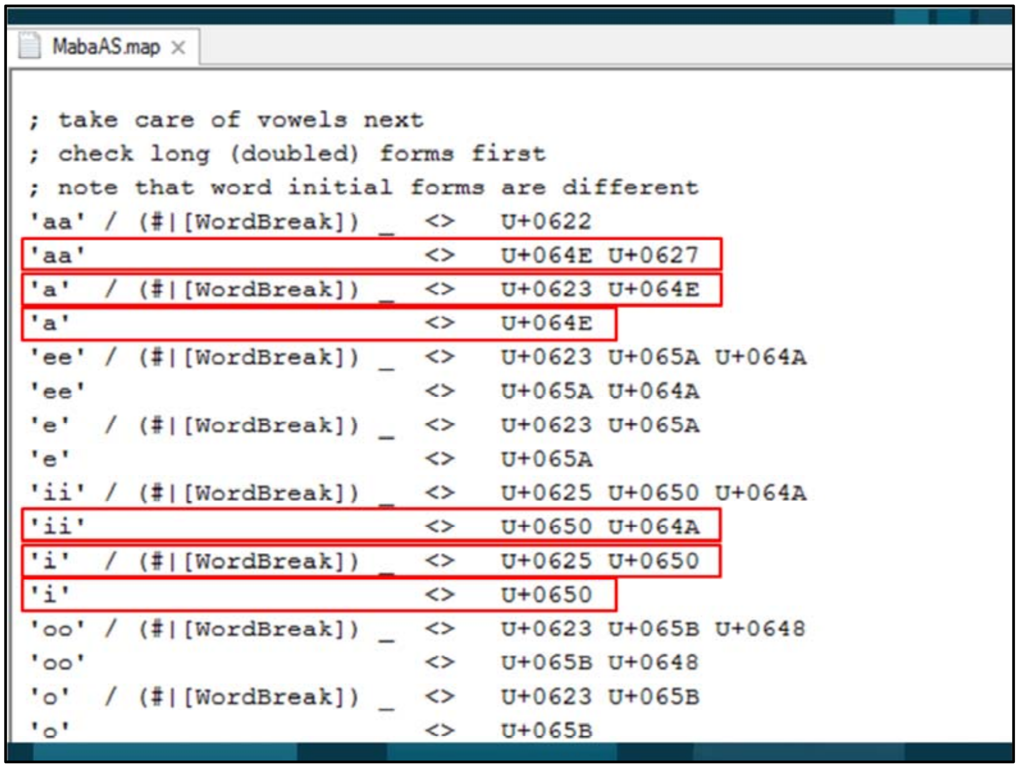
How are we doing?

OK, time for a process check. Would someone like to try to summarize what we've seen so far?
Unicode Arabic Script code page is your friend – to find characters you need to convert to
TECkit mapping file is a magical thing which can convert between RS and AS
It is unique to each language
Any questions?



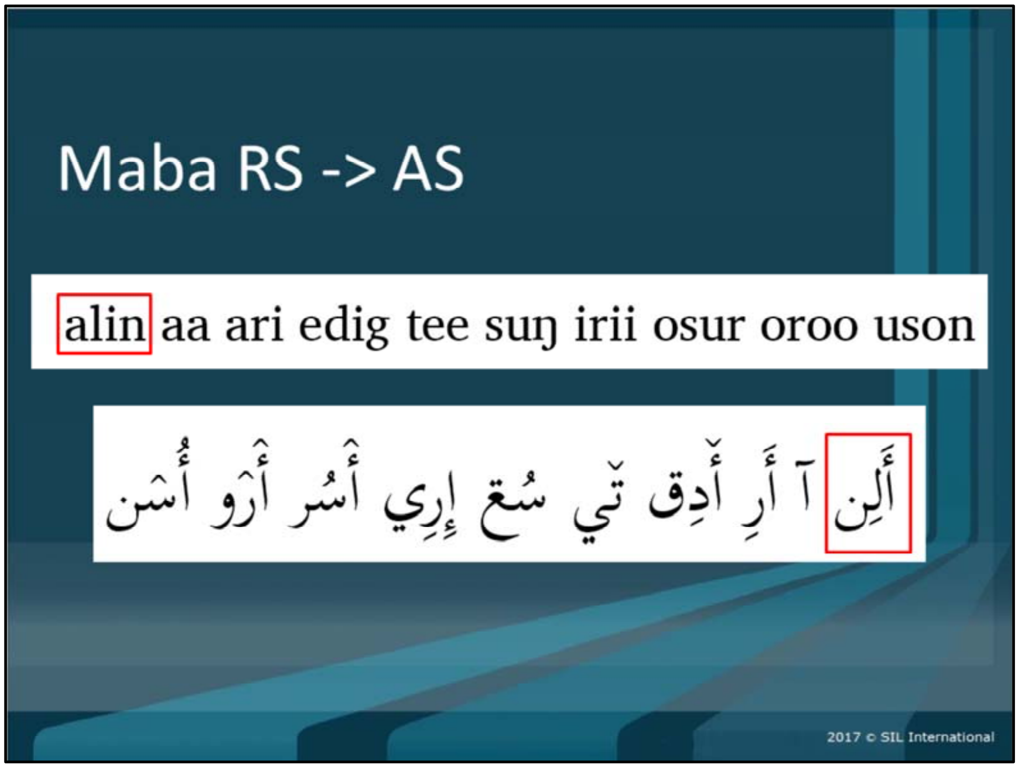
```
; consonants
'b'          <>   U+0628
'p'          <>   U+067E
't'          <>   U+062A
'j'          <>   U+062C
'nj'         <>   U+0683
'c'          <>   U+0686
'd'          <>   U+062F
'nd'         <>   U+068A
'r'          <>   U+0631
'z'          <>   U+0632
'dr'         <>   U+0694
'tr'         <>   U+0697
'ndr'        <>   U+0698
's'          <>   U+0633
'ch'         <>   U+0634
U+014B       <>   U+075D   ; eng
U+014B 'g'   <>   U+06A0   ; eng-g
'f'          <>   U+0641
'g'          <>   U+0642
```

All we've seen so far in our TECkit mapping file is some simple replacements of consonants or sequences of consonants. That's the simplest kind of conversion – more or less a one-to-one mapping.


MabaAS.map

```
; take care of vowels next
; check long (doubled) forms first
; note that word initial forms are different
'aa' / (#|[WordBreak]) _   <>   U+0622
'aa'                       <>   U+064E U+0627
'a'  / (#|[WordBreak]) _   <>   U+0623 U+064E
'a'                        <>   U+064E
'ee' / (#|[WordBreak]) _   <>   U+0623 U+065A U+064A
'ee'                       <>   U+065A U+064A
'e'  / (#|[WordBreak]) _   <>   U+0623 U+065A
'e'                        <>   U+065A
'ii' / (#|[WordBreak]) _   <>   U+0625 U+0650 U+064A
'ii'                       <>   U+0650 U+064A
'i'  / (#|[WordBreak]) _   <>   U+0625 U+0650
'i'                        <>   U+0650
'oo' / (#|[WordBreak]) _   <>   U+0623 U+065B U+0648
'oo'                       <>   U+065B U+0648
'o'  / (#|[WordBreak]) _   <>   U+0623 U+065B
'o'                        <>   U+065B
```

But there are some more sophisticated things you can do in your mapping file as well. Let’s look at a little more complex section – the section for converting vowels.
□ There are some mappings here that are the same as what we’ve already seen, like the mapping between “a” and fatha.
□ And the mappings on the double vowels is similar. A double “a” just goes to a fatha followed by an Alif.
□ But what are **these** rules saying? What do you think this slash followed by some other stuff is? Context – probably not a foreign notation to those of you who have done much linguistics. It says that an ‘a’ in a particular context changes to a 0623 and a 064E (an alef hamza and a fatha), instead of a simple 064E (a fatha). And what context is that? The context where that ‘a’ appears after a WordBreak character or at the beginning of the text, which is represented by the number sign. The underscore shows the place where the character to convert would have to appear. So a word initial ‘a’ gets converted to an alef hamza and a fatha.



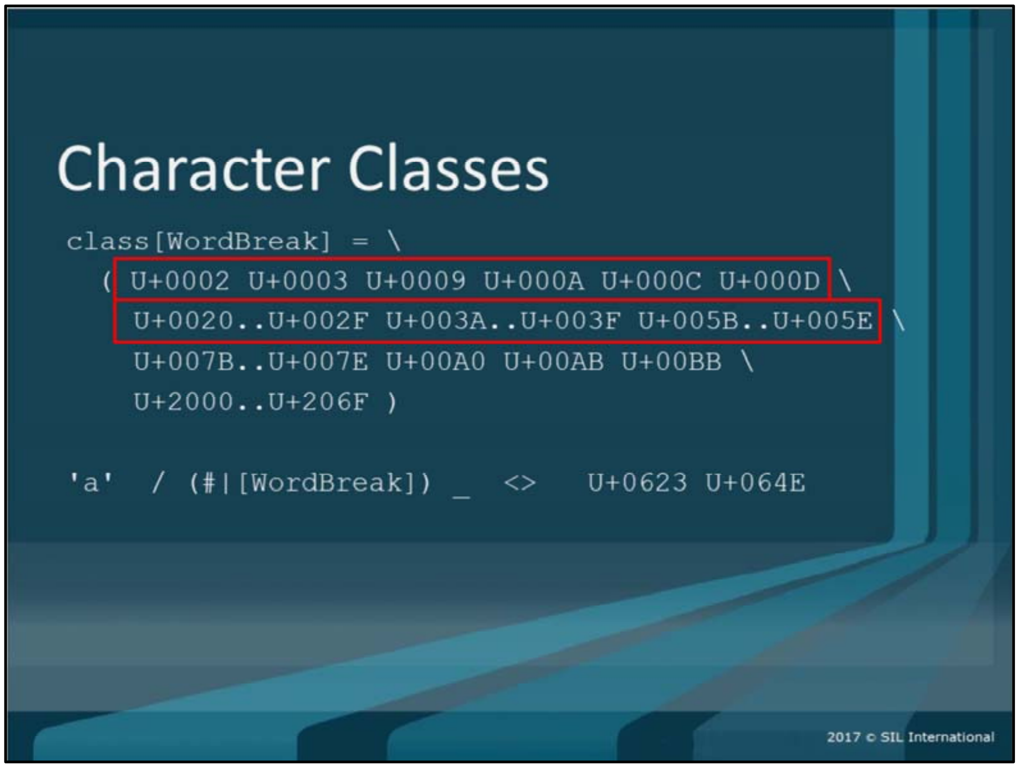
Maba RS -> AS

alin aa ari edig tee suŋ irii osur oroo uson

أَلِن آ أَرِ أ̌دِق ت̌ي سُغ إِرِي أ̂سُر أر̂و أُس̂ن

2017 © SIL International

And we can verify that in our sample Maba text. The first word begins with an ‘a’, so it was converted to an alef hamza and a fatha.



Character Classes

```
class[WordBreak] = \
  ( U+0002 U+0003 U+0009 U+000A U+000C U+000D \
    U+0020..U+002F U+003A..U+003F U+005B..U+005E \
    U+007B..U+007E U+00A0 U+00AB U+00BB \
    U+2000..U+206F )

'a'  / (#|[WordBreak]) _  <>   U+0623 U+064E
```

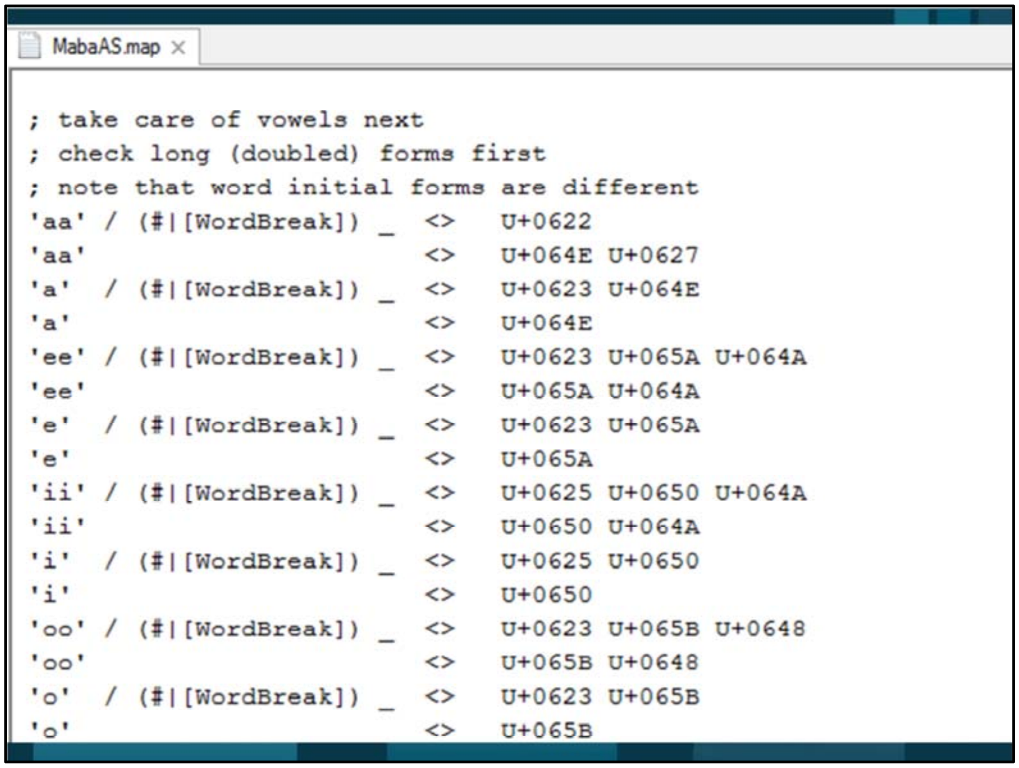
2017 © SIL International

What I didn't show you yet is the definition of the WordBreak class. Character classes may be used to make the mapping description more readable and concise. There are a lot of different characters which could break words apart – some control characters, punctuation, and spaces. So we try to make a list of all of the word breaking characters that we might see.

□ Some characters are just individual characters, like the 6 characters defined in this first line. (The backslash just means that the mapping rule continues onto the next line.)

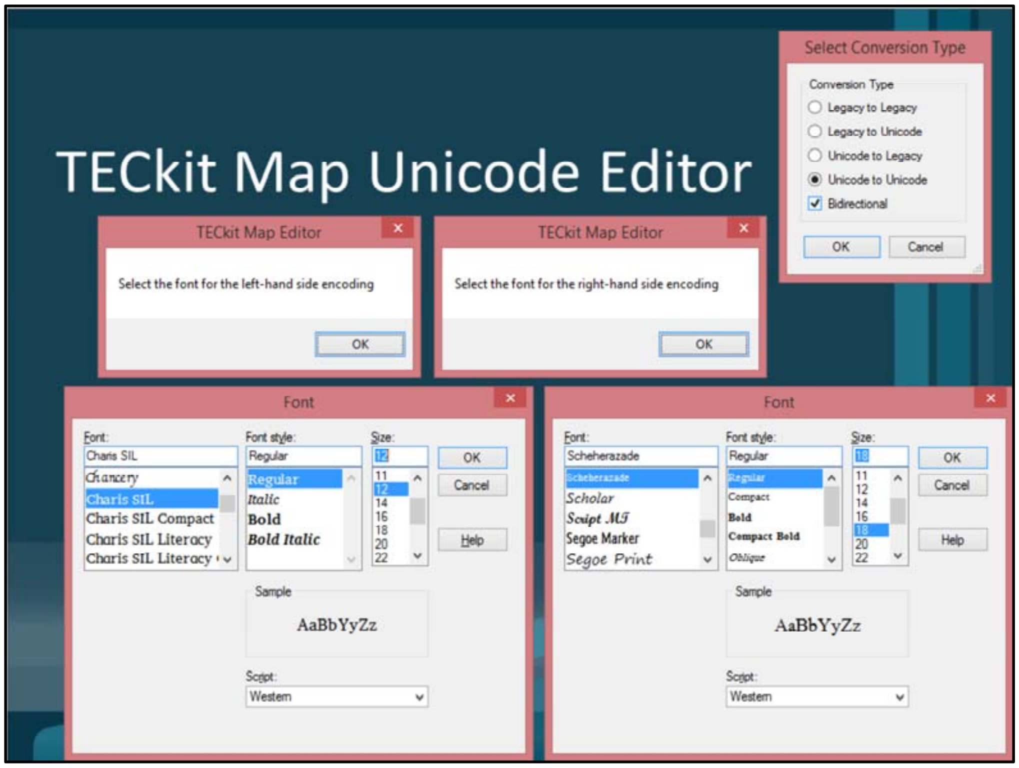
□ But the second line contains 3 character ranges. The double-period indicates that it includes all characters from the first one given to the last. So that first range includes all characters from 0020 hexadecimal up to 002F hexadecimal. How many characters is that? It's 16 characters that it adds to the character class.

□ Sometimes building up these character classes might involve a little bit of trial and error. You put in what you think you need, and later on, you might convert a text and realize that there is a word-breaking character which is not forcing the vowel following it to take its word-initial form. In that case, you figure out its Unicode code and add it to your word-breaking character list.



```
; take care of vowels next
; check long (doubled) forms first
; note that word initial forms are different
'aa' / (#|[WordBreak]) _  <>   U+0622
'aa'                      <>   U+064E U+0627
'a'  / (#|[WordBreak]) _  <>   U+0623 U+064E
'a'                       <>   U+064E
'ee' / (#|[WordBreak]) _  <>   U+0623 U+065A U+064A
'ee'                      <>   U+065A U+064A
'e'  / (#|[WordBreak]) _  <>   U+0623 U+065A
'e'                       <>   U+065A
'ii' / (#|[WordBreak]) _  <>   U+0625 U+0650 U+064A
'ii'                      <>   U+0650 U+064A
'i'  / (#|[WordBreak]) _  <>   U+0625 U+0650
'i'                       <>   U+0650
'oo' / (#|[WordBreak]) _  <>   U+0623 U+065B U+0648
'oo'                      <>   U+065B U+0648
'o'  / (#|[WordBreak]) _  <>   U+0623 U+065B
'o'                       <>   U+065B
```

So this is pretty cool, to be able to convert things differently depending on their context. Give me some examples of how that could be useful. What other contexts might be important?
Word initial, word final (e.g. suffixes), word entire, not followed by a certain class of characters (e.g. vowel), consonant “colored” by a certain type of vowel, punctuation surrounded by numbers



TECkit Map Unicode Editor

Select Conversion Type

Conversion Type
- Legacy to Legacy
- Legacy to Unicode
- Unicode to Legacy
- [✓ Unicode to Unicode]
- [✓ Bidirectional]

TECkit Map Editor: Select the font for the left-hand side encoding

TECkit Map Editor: Select the font for the right-hand side encoding

Font: Charis SIL — Font style: Regular — Size: 12 — Sample: AaBbYyZz — Script: Western

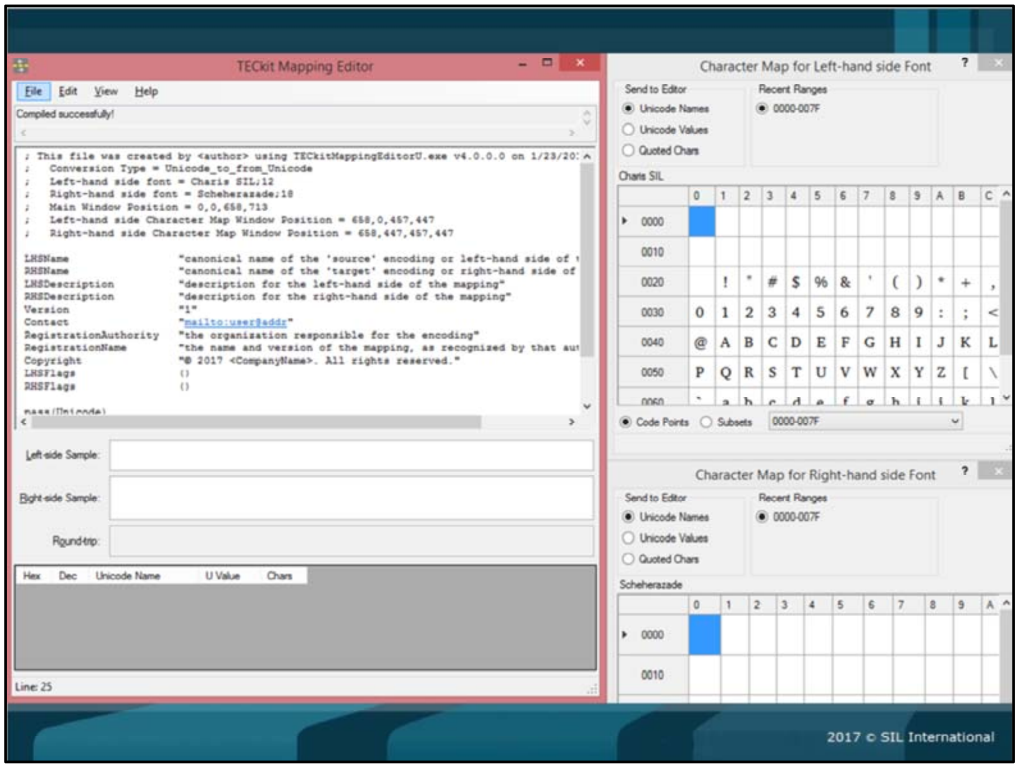
Font: Scheherazade — Font style: Regular — Size: 18 — Sample: AaBbYyZz — Script: Western

We'll be talking more about TECkit mapping files, but before we go any further, I wanted to show you a tool that you might find helpful for editing your mapping files. Up to now I've just shown you raw mapping files in a text editor. That's often the way I work, but there is also a tool called the TECkit Map Unicode Editor which is installed with SIL Converters that you might appreciate using.

□ When you first start the program there are a number of dialog boxes that come up, first to select the type of conversion – I would recommend selecting Unicode to Unicode and bidirectional.

□ Then you select a font for the left-hand side encoding, in other words our Roman script, so Charis SIL is a good choice

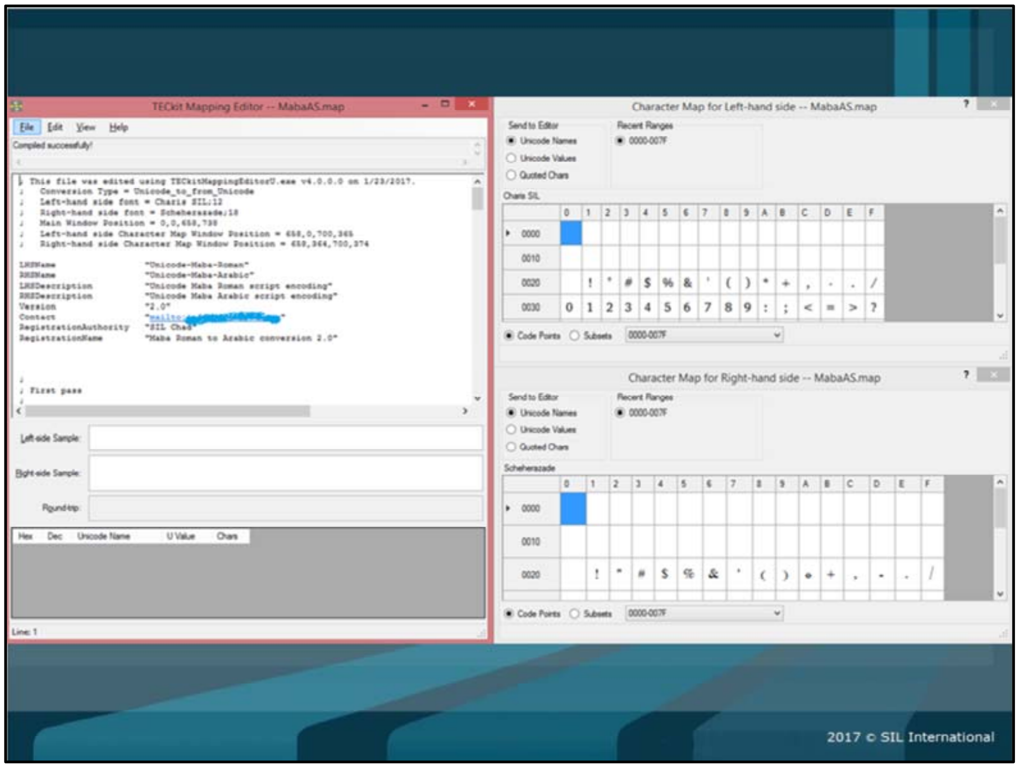
□ and then you select a font for the right-hand side encoding, that is for our Arabic script, so Scheherazade is a good choice.



Compiled successfully!

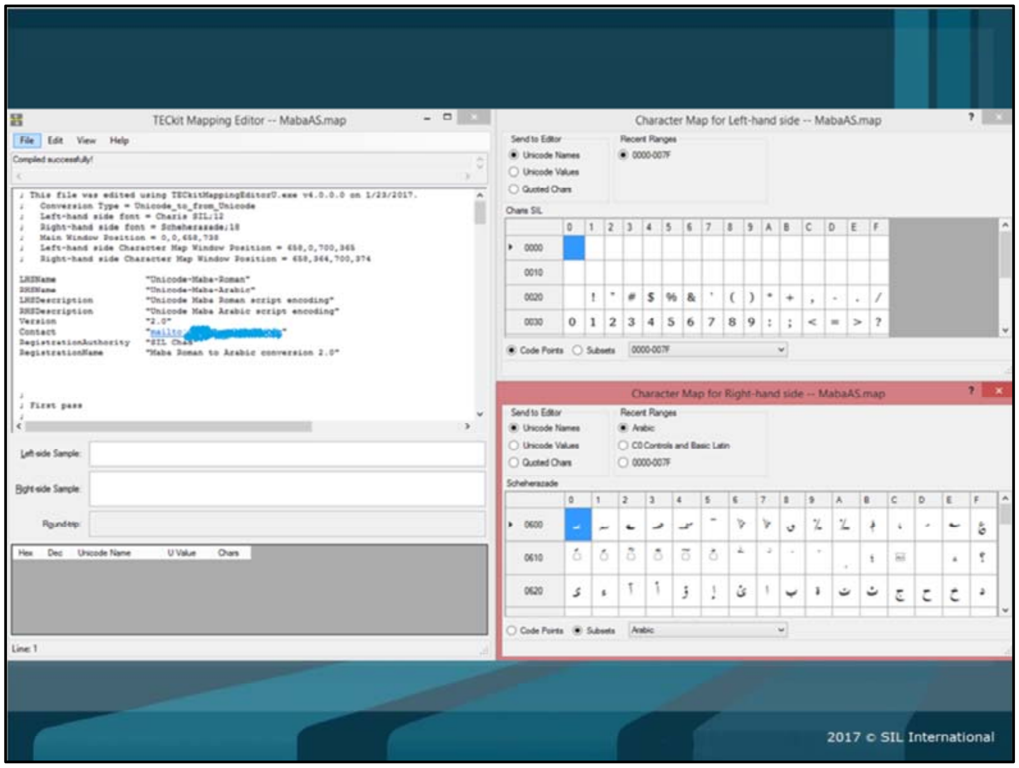
```
; This file was created by <author> using TECkitMappingEditorU.exe v4.0.0.0 on 1/23/20:
;   Conversion Type = Unicode_to_from_Unicode
;   Left-hand side font = Charis SIL;12
;   Right-hand side font = Scheherazade;18
;   Main Window Position = 0,0,658,713
;   Left-hand side Character Map Window Position = 658,0,457,447
;   Right-hand side Character Map Window Position = 658,447,457,447

LHSName                 "canonical name of the 'source' encoding or left-hand side of t
RHSName                 "canonical name of the 'target' encoding or right-hand side of
LHSDescription          "description for the left-hand side of the mapping"
RHSDescription          "description for the right-hand side of the mapping"
Version                 "1"
Contact                 "mailto:user@addr"
RegistrationAuthority   "the organization responsible for the encoding"
RegistrationName        "the name and version of the mapping, as recognized by that aut
Copyright               "© 2017 <CompanyName>. All rights reserved."
LHSFlags                ()
RHSFlags                ()

pass(Unicode)
```

Then it comes up with a small example conversion mapping file, and a couple of windows for selecting characters. If you already have a mapping file, you can use the File menu to open it. You can also set this program as the default program to open your mapping files, which we usually give a .map extension.



```
; This file was edited using TECkitMappingEditorU.exe v4.0.0.0 on 1/23/2017.
;   Conversion Type = Unicode_to_from_Unicode
;   Left-hand side font = Charis SIL;12
;   Right-hand side font = Scheherazade;18
;   Main Window Position = 0,0,658,738
;   Left-hand side Character Map Window Position = 658,0,700,365
;   Right-hand side Character Map Window Position = 658,364,700,374

LHSName                 "Unicode-Maba-Roman"
RHSName                 "Unicode-Maba-Arabic"
LHSDescription          "Unicode Maba Roman script encoding"
RHSDescription          "Unicode Maba Arabic script encoding"
Version                 "2.0"
Contact                 "mailto:[redacted]"
RegistrationAuthority   "SIL Chad"
RegistrationName        "Maba Roman to Arabic conversion 2.0"


;
; First pass
;
```

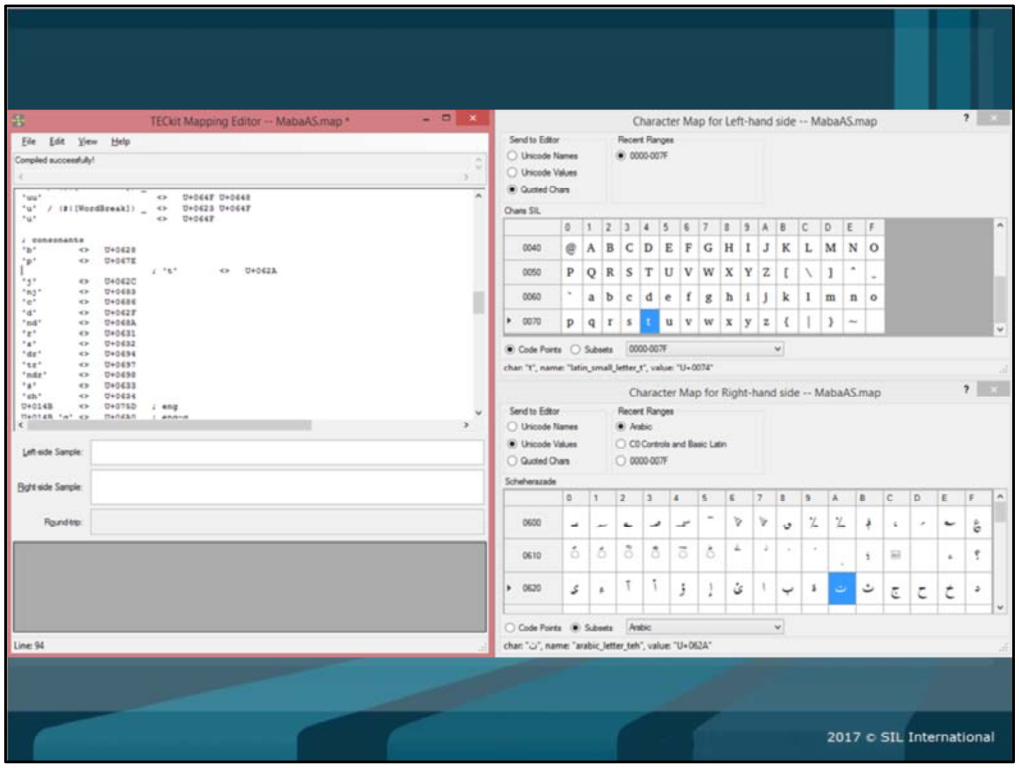
So if I double-click on the MabaAS.map file, it opens up our Maba mapping file, ready for editing. In addition to the editing pane, there is a section where you can test your mapping file, and two separate windows to the right with character maps. The editor puts information about the fonts and window positions as comments at the beginning of the mapping file, so you don't have to go through the select of the fonts and positioning of the windows each time, which gets old real fast!
One of the first things we want to do is to position the character map for the Right-hand-side on the Arabic code page. You can do that by selecting 0600 in the drop-down list, or by selecting the Subsets radio button and then selecting Arabic in the drop-down list.



```
; This file was edited using TECkitMappingEditorU.exe v4.0.0.0 on 1/23/2017.
;   Conversion Type = Unicode_to_from_Unicode
;   Left-hand side font = Charis SIL;12
;   Right-hand side font = Scheherazade;18
;   Main Window Position = 0,0,658,738
;   Left-hand side Character Map Window Position = 658,0,700,365
;   Right-hand side Character Map Window Position = 658,364,700,374

LHSName                 "Unicode-Maba-Roman"
RHSName                 "Unicode-Maba-Arabic"
LHSDescription          "Unicode Maba Roman script encoding"
RHSDescription          "Unicode Maba Arabic script encoding"
Version                 "2.0"
Contact                 "mailto:[redacted]"
RegistrationAuthority   "SIL Cha[redacted]
RegistrationName        "Maba Roman to Arabic conversion 2.0"


;
; First pass
;
```

Compiled successfully!

To see one of the ways this mapping editor can help me, let's remove the line that converts "t" to "TEH", and show how we can put it back in.



TECkit Mapping Editor -- MabaAS.map *

Compiled successfully!

```
'uu'                          <>   U+064F U+0648
'u'  / (#|[WordBreak]) _      <>   U+0623 U+064F
'u'                           <>   U+064F

; consonants
'b'        <>   U+0628
'p'        <>   U+067E
                      ; 't'        <>   U+062A
'j'        <>   U+062C
'nj'       <>   U+0683
'c'        <>   U+0686
'd'        <>   U+062F
'nd'       <>   U+068A
'r'        <>   U+0631
'z'        <>   U+0632
'dz'       <>   U+0694
'tz'       <>   U+0697
'ndz'      <>   U+0698
's'        <>   U+0633
'sh'       <>   U+0634
U+014B     <>   U+075D   ; eng
```

Line: 94

Character Map for Left-hand side -- MabaAS.map

Send to Editor: Unicode Names ○ Unicode Values ○ [✓ Quoted Chars]
Recent Ranges: [✓ 0000-007F]

char: "t", name: "latin_small_letter_t", value: "U+0074"

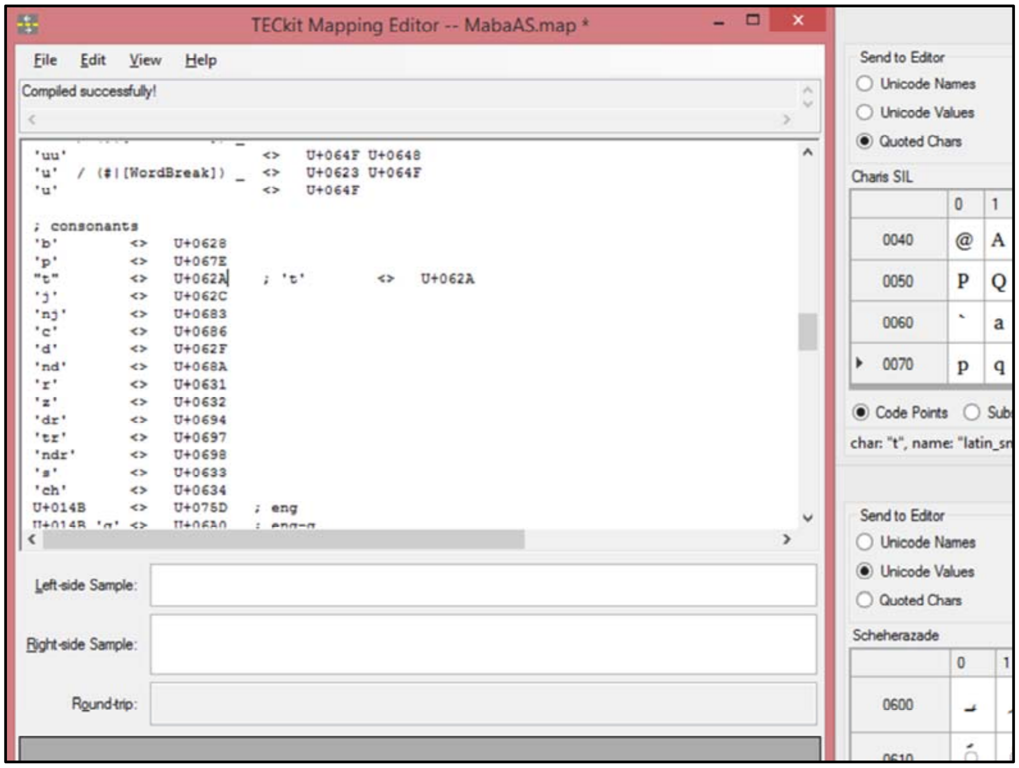
Character Map for Right-hand side -- MabaAS.map

Send to Editor: Unicode Names ○ [✓ Unicode Values] Quoted Chars ○
Recent Ranges: [✓ Arabic] C0 Controls and Basic Latin ○ 0000-007F ○

char: "ت", name: "arabic_letter_teh", value: "U+062A"

2017 © SIL International

We put the insertion point where we want to add the mapping rule, then we use the windows to the right to select what we want to insert in our mapping file. We select "Quoted Chars" for the Left-hand-side and click the "t", then tab or space over and type "<>", space over, then select "Unicode Values" for the Right-hand-side and click on the "TEH".

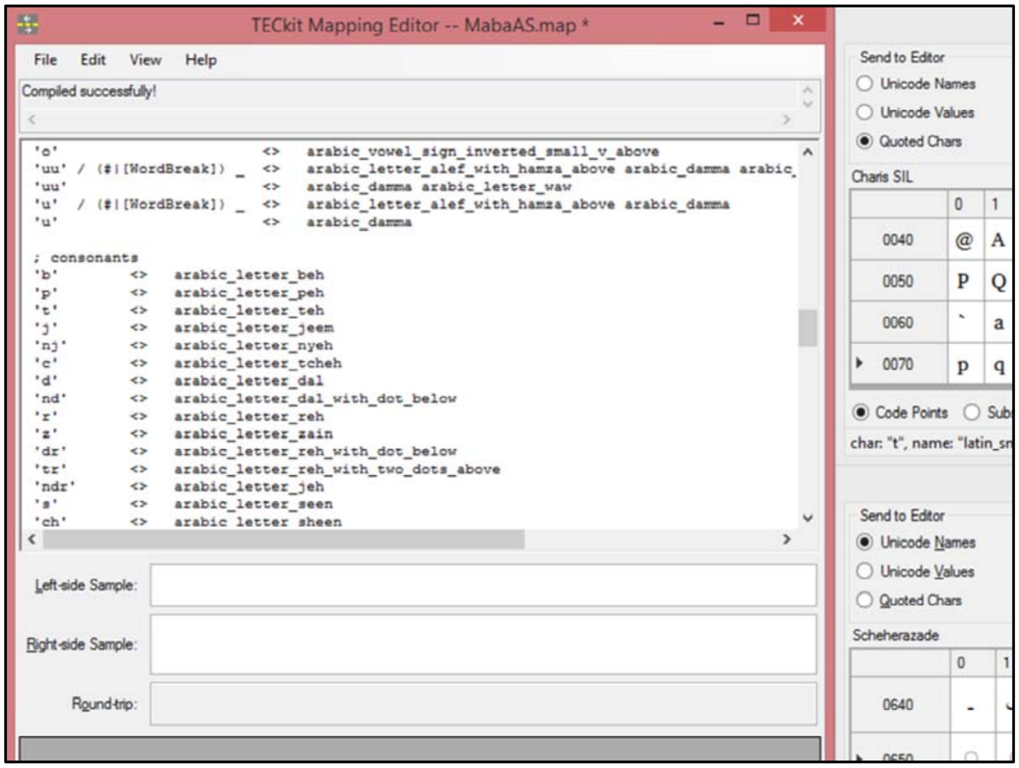


Compiled successfully!

```
'uu'                         <>   U+064F U+0648
'u'  / (#|[WordBreak]) _     <>   U+0623 U+064F
'u'                          <>   U+064F

; consonants
'b'          <>   U+0628
'p'          <>   U+067E
"t"          <>   U+062A    ; 't'          <>   U+062A
'j'          <>   U+062C
'nj'         <>   U+0683
'c'          <>   U+0686
'd'          <>   U+062F
'nd'         <>   U+068A
'r'          <>   U+0631
'z'          <>   U+0632
'dr'         <>   U+0694
'tr'         <>   U+0697
'ndr'        <>   U+0698
's'          <>   U+0633
'ch'         <>   U+0634
U+014B       <>   U+075D    ; eng
```

char: "t", name: "latin_sm

That re-inserts our mapping rule, without having to look up codes in the Unicode chart. It did use a different kind of quotes, but either kind can be used – they have the same meaning.



Compiled successfully!

```
'o'                        <>   arabic_vowel_sign_inverted_small_v_above
'uu' / (#|[WordBreak]) _   <>   arabic_letter_alef_with_hamza_above arabic_damma arabic_
'uu'                       <>   arabic_damma arabic_letter_waw
'u'  / (#|[WordBreak]) _   <>   arabic_letter_alef_with_hamza_above arabic_damma
'u'                        <>   arabic_damma

; consonants
'b'        <>   arabic_letter_beh
'p'        <>   arabic_letter_peh
't'        <>   arabic_letter_teh
'j'        <>   arabic_letter_jeem
'nj'       <>   arabic_letter_nyeh
'c'        <>   arabic_letter_tcheh
'd'        <>   arabic_letter_dal
'nd'       <>   arabic_letter_dal_with_dot_below
'r'        <>   arabic_letter_reh
'z'        <>   arabic_letter_zain
'dr'       <>   arabic_letter_reh_with_dot_below
'tr'       <>   arabic_letter_reh_with_two_dots_above
'ndr'      <>   arabic_letter_jeh
's'        <>   arabic_letter_seen
'ch'       <>   arabic_letter_sheen
```

Send to Editor: ( ) Unicode Names ( ) Unicode Values [✓ Quoted Chars]

Charis SIL

	0	1
0040	@	A
0050	P	Q
0060	`	a
0070	p	q

[✓ Code Points]

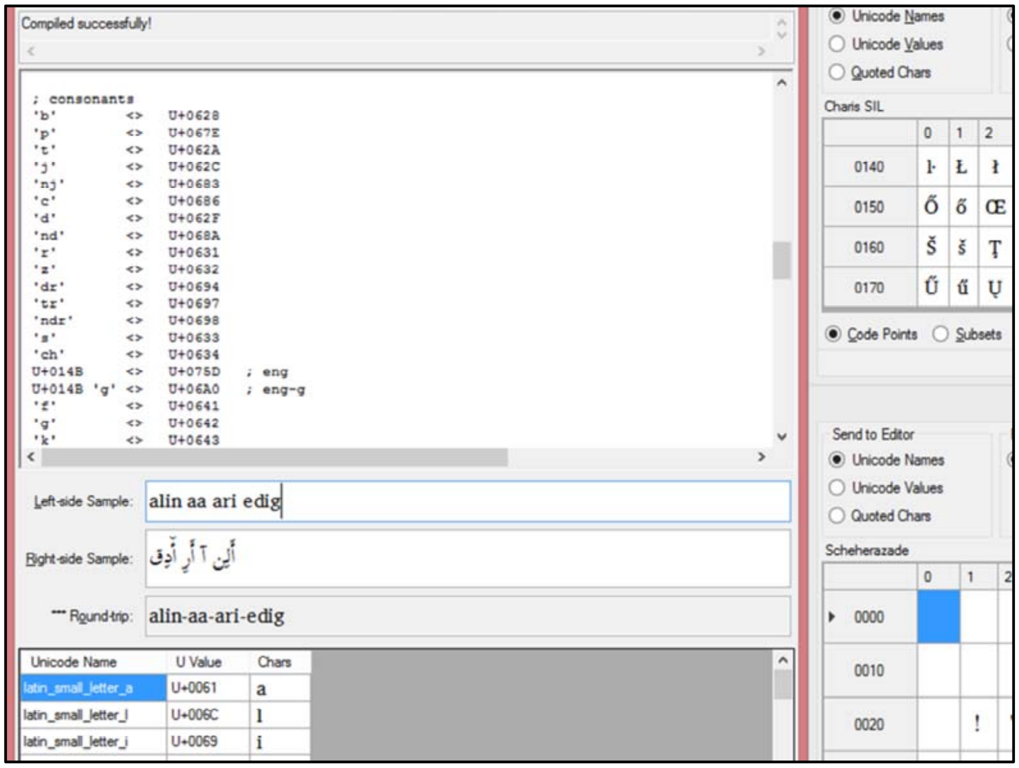
char: "t", name: "latin_sm

Send to Editor: [✓ Unicode Names] ( ) Unicode Values ( ) Quoted Chars

Scheherazade

	0	1
0640	-	

In addition to inserting characters as Unicode values or as quoted strings, you can also format them as Unicode Names. Some people find that mapping files created like this are more self-documenting. SIL Converters handles all of these different character formats, and they can even be used interchangeably in mapping files – maybe using the longer Unicode Names where you particularly want to make clear what the mapping file is doing.
What does the semicolon mean again? The beginning of a comment.
If you look at something a while later and don't remember what you were doing, that means you should have put a comment there! Just type a semicolon followed by some text that helps you remember what is going on.


Compiled successfully!

```
; consonants
'b'         <>   U+0628
'p'         <>   U+067E
't'         <>   U+062A
'j'         <>   U+062C
'nj'        <>   U+0683
'c'         <>   U+0686
'd'         <>   U+062F
'nd'        <>   U+068A
'r'         <>   U+0631
'z'         <>   U+0632
'dr'        <>   U+0694
'tr'        <>   U+0697
'ndr'       <>   U+0698
's'         <>   U+0633
'ch'        <>   U+0634
U+014B      <>   U+075D   ; eng
U+014B 'g'  <>   U+06A0   ; eng-g
'f'         <>   U+0641
'g'         <>   U+0642
'k'         <>   U+0643
```

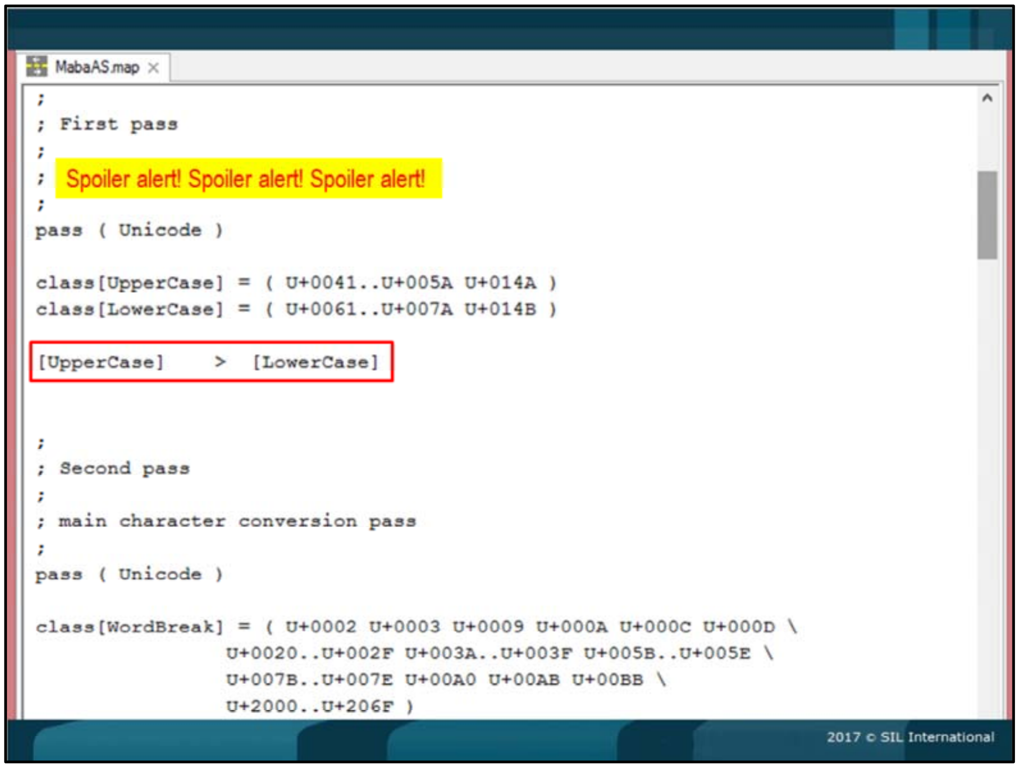
Left-side Sample: alin aa ari edig

Right-side Sample: ألِن آ أرِ أدِق

*** Round-trip: alin-aa-ari-edig

Unicode Name	U Value	Chars
latin_small_letter_a	U+0061	a
latin_small_letter_l	U+006C	l
latin_small_letter_i	U+0069	i

Before I go on, I wanted to draw your attention to something that many people appreciate about the TECkit Map Unicode Editor. You can type or paste a sample of text in the Left-side Sample box, and it will run it through the conversion for you, and even try to run the conversion backwards from there, giving you the resulting Round-trip text. And if you click in one of the sample boxes, it gives you a list below of all of the characters with their codes and Unicode names. The program default is that your mapping file is continually auto-compiled – see that "Compiled successfully!" message at the top? So as you change your mapping file, you can see the effects of those changes immediately in the sample boxes. And if your mapping file can't compile, it will give you an error message up there. If you double-click on that error message, it will highlight the line with the error.
You can see in this case that the Round-trip result is not the same as the original – it has hyphens instead of spaces. But MabaAS was designed as a Roman to Arabic script converter, so it's not too surprising that it doesn't work exactly right if we run it backwards. This ability to get immediate feedback on how your mapping file works is a Very Good Thing!

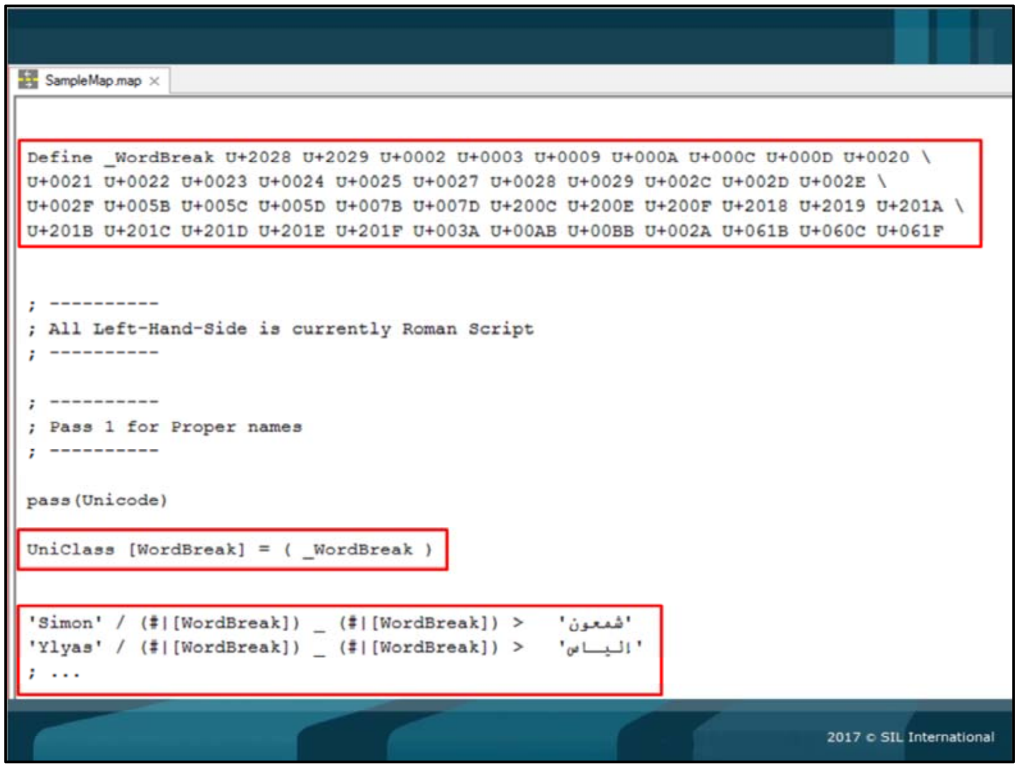


MabaAS.map

```
;
; First pass
;
; Spoiler alert! Spoiler alert! Spoiler alert!
;
pass ( Unicode )

class[UpperCase] = ( U+0041..U+005A U+014A )
class[LowerCase] = ( U+0061..U+007A U+014B )

[UpperCase]     >  [LowerCase]


;
; Second pass
;
; main character conversion pass
;
pass ( Unicode )

class[WordBreak] = ( U+0002 U+0003 U+0009 U+000A U+000C U+000D \
                U+0020..U+002F U+003A..U+003F U+005B..U+005E \
                U+007B..U+007E U+00A0 U+00AB U+00BB \
                U+2000..U+206F )
```

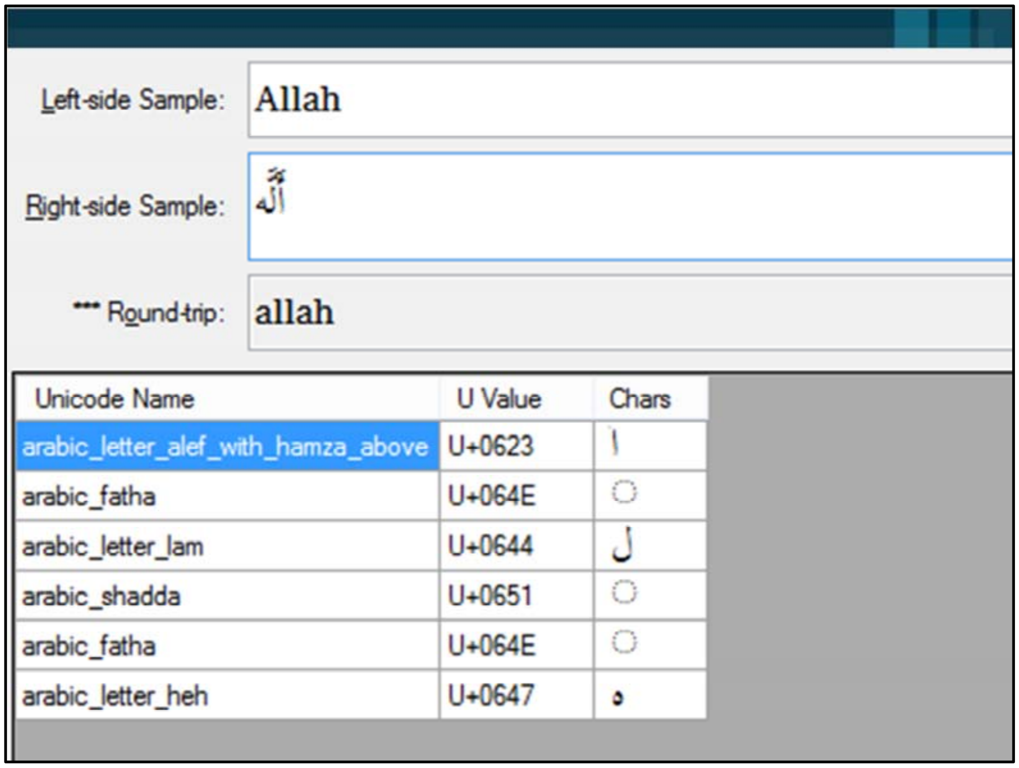
2017 © SIL International

Let me quickly go through a few more commands and techniques that are helpful to use in mapping files. One command that is very helpful is the “pass” command. This allows you to define distinct conversion passes – an earlier conversion pass is carried through to completion before the next conversion pass is started. That allows your conversion work to be compartmentalized so that things don’t get quite so messy.
The portion of the mapping file here has two separate passes. The second one (with just a little bit of it showing) says that it is the main character conversion pass, which converts the bulk of characters from Roman to Arabic script.
Take a look at the code in the first pass… Can you tell me what it does?
□ It converts all Roman script characters to lowercase. Can anyone explain what these character classes are and how they are defined?
A: 0041..005A is A-Z, 014A is capital Eng, same for lowercase
If we change everything to lowercase, then we can just look for a lowercase “b” to convert to “BEH” in the next pass, rather than looking for both upper and lowercase. So it makes the other passes a lot simpler.
□ What about this conversion rule… Does anyone notice anything different? It only goes one way!
That’s because we can happily convert to lowercase when producing Arabic script, but if we are going to reverse the conversion, there is no way to know which of the resulting Roman script characters should be capitalized. We could write some context rules that could capitalize the first word of a text, or a word following a sentence final punctuation. But what about proper names in the middle of a sentence? It might be more challenging to restore those capital letters.



```
Define _WordBreak U+2028 U+2029 U+0002 U+0003 U+0009 U+000A U+000C U+000D U+0020 \
U+0021 U+0022 U+0023 U+0024 U+0025 U+0027 U+0028 U+0029 U+002C U+002D U+002E \
U+002F U+005B U+005C U+005D U+007B U+007D U+200C U+200E U+200F U+2018 U+2019 U+201A \
U+201B U+201C U+201D U+201E U+201F U+003A U+00AB U+00BB U+002A U+061B U+060C U+061F


; ----------
; All Left-Hand-Side is currently Roman Script
; ----------

; ----------
; Pass 1 for Proper names
; ----------

pass(Unicode)

UniClass [WordBreak] = ( _WordBreak )

'Simon' / (#|[WordBreak]) _ (#|[WordBreak]) >   'شمعون'
'Ylyas' / (#|[WordBreak]) _ (#|[WordBreak]) >   'الـيـــاس'
; ...
```

While we are talking about proper names, let me show you a portion of the SampleMap mapping file that is available on the scripts.sil.org website.
□ First I will note that it is possible to define symbols with a macro, like this _WordBreak symbol that is equivalent to a long list of word breaking characters.
□ That symbol can then be used in a very simple WordBreak class definition. Classes need to be defined in the pass in which they are used, so if the WordBreak class needs to be used in more than one pass, then defining that symbol with a macro will save us a lot of duplication.
□ Now look at this pass for proper names.
□ How would you read those conversion rules?
If you find "Simon" in the context where it is preceded and followed by word break characters, then convert it to that AS string.
Names quite often have some idiosyncrasies in the way they are written, so converting those special forms in their entirety is usually a good idea.



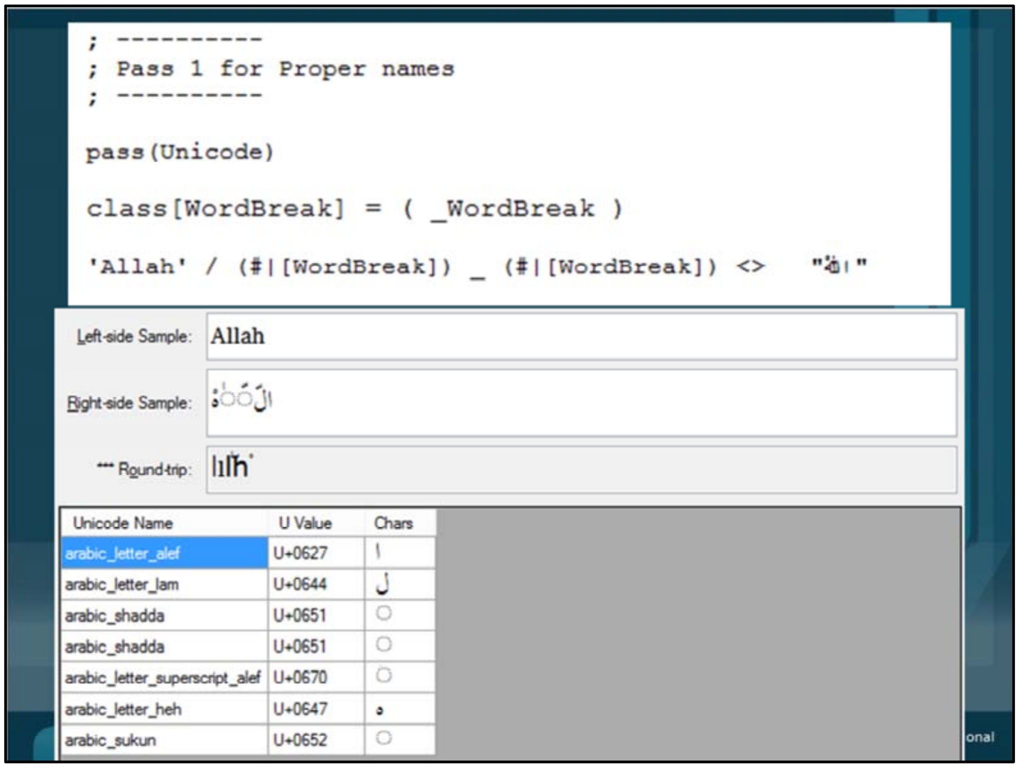
Left-side Sample: Allah

Right-side Sample: أَلَّه

*** Round-trip: allah

Unicode Name	U Value	Chars
arabic_letter_alef_with_hamza_above	U+0623	أ
arabic_fatha	U+064E	◌َ
arabic_letter_lam	U+0644	ل
arabic_shadda	U+0651	◌ّ
arabic_fatha	U+064E	◌َ
arabic_letter_heh	U+0647	ه

Allah is a good case in point. Remember before that Allah wasn't coming out the way we wanted it in Arabic script? The normal Maba mapping table makes it come out like this.



```
; ----------
; Pass 1 for Proper names
; ----------

pass(Unicode)

class[WordBreak] = ( _WordBreak )

'Allah' / (#|[WordBreak]) _ (#|[WordBreak]) <>   "ﷲا"
```

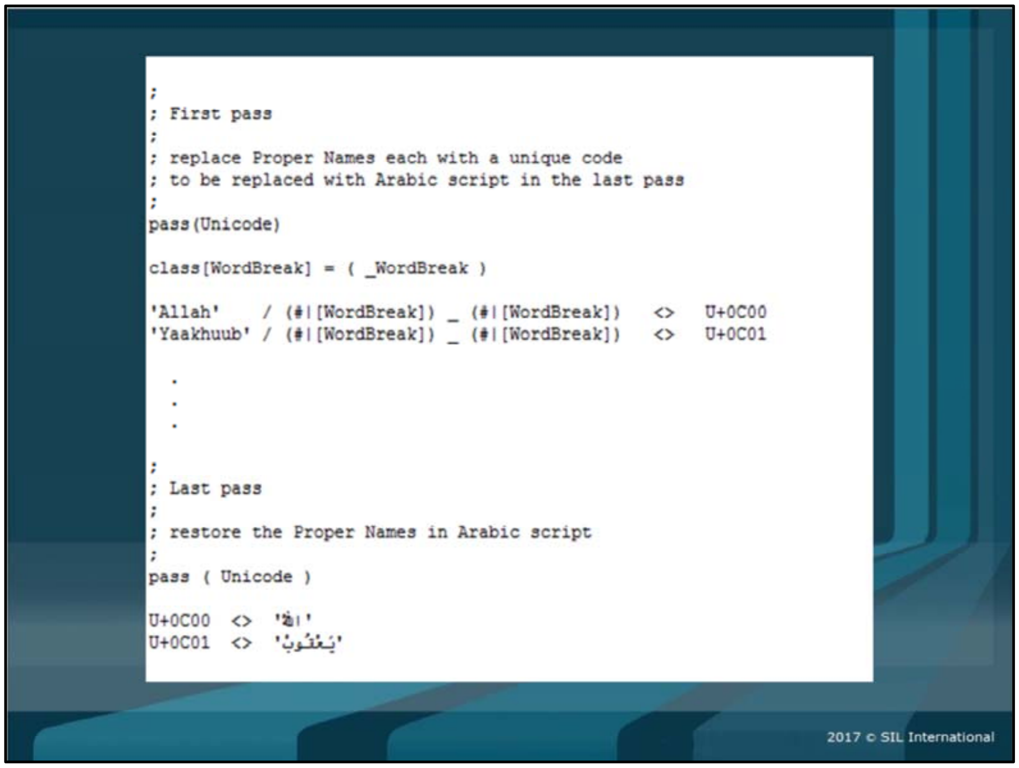
Left-side Sample: Allah

Right-side Sample: الّّٰهْ

Round-trip: lılh˙

Unicode Name	U Value	Chars
arabic_letter_alef	U+0627	ا
arabic_letter_lam	U+0644	ل
arabic_shadda	U+0651	◌
arabic_shadda	U+0651	◌
arabic_letter_superscript_alef	U+0670	◌
arabic_letter_heh	U+0647	ه
arabic_sukun	U+0652	◌

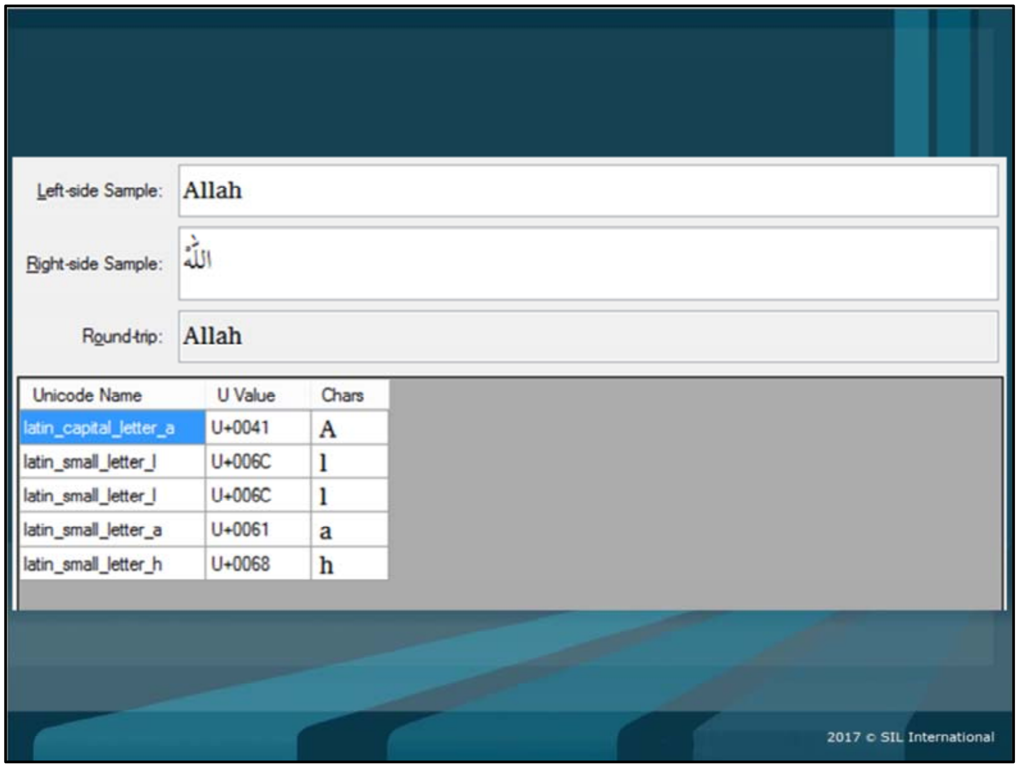
So you might think that something like this would solve your problem. You can just trust me that the squiggle between the quotes is what we really want for Allah.
□ Unfortunately, when we test it out, we *<b>don’t</b>* get what we want. The problem is that what we want contains two consecutive LAM characters followed by a shadda, but we have a pass later on that converts two consecutive consonants to a consonant and a shadda.



```
;
; First pass
;
; replace Proper Names each with a unique code
; to be replaced with Arabic script in the last pass
;
pass(Unicode)

class[WordBreak] = ( _WordBreak )

'Allah'    / (#|[WordBreak]) _ (#|[WordBreak])   <>   U+0C00
'Yaakhuub' / (#|[WordBreak]) _ (#|[WordBreak])   <>   U+0C01

  .
  .
  .

;
; Last pass
;
; restore the Proper Names in Arabic script
;
pass ( Unicode )

U+0C00  <>  'الله'
U+0C01  <>  'يَعْقُوبْ'
```

2017 © SIL International

So there is a different kind of technique we'll use instead. We change each of our Proper Names to some unique code in the very first pass – it doesn't really matter what codes we use as long as they are codes that doesn't appear anywhere in your texts. The 0C00 code page is for the Telugu script, which I think is pretty safe for most of us! So in our first pass, we change each Proper Name to a unique code, finish the rest of the conversion, and then for the last pass, we change each of those unique codes back into the proper Arabic script form of the Proper Name.



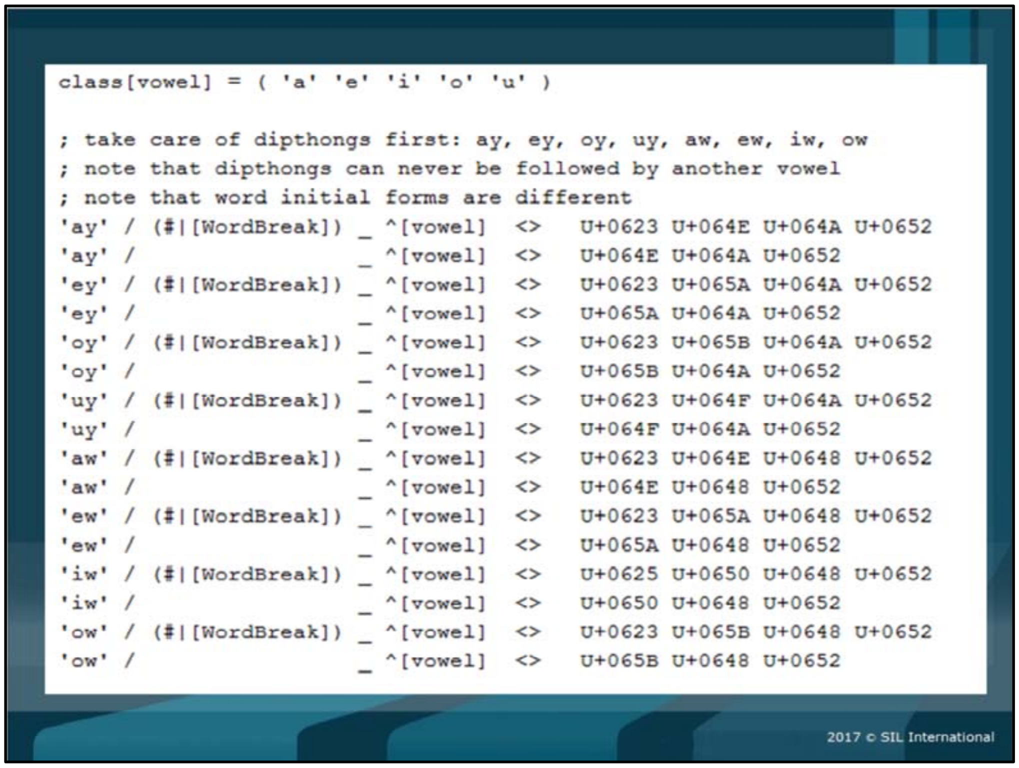
Left-side Sample: Allah

Right-side Sample: ﷲ

Round-trip: Allah

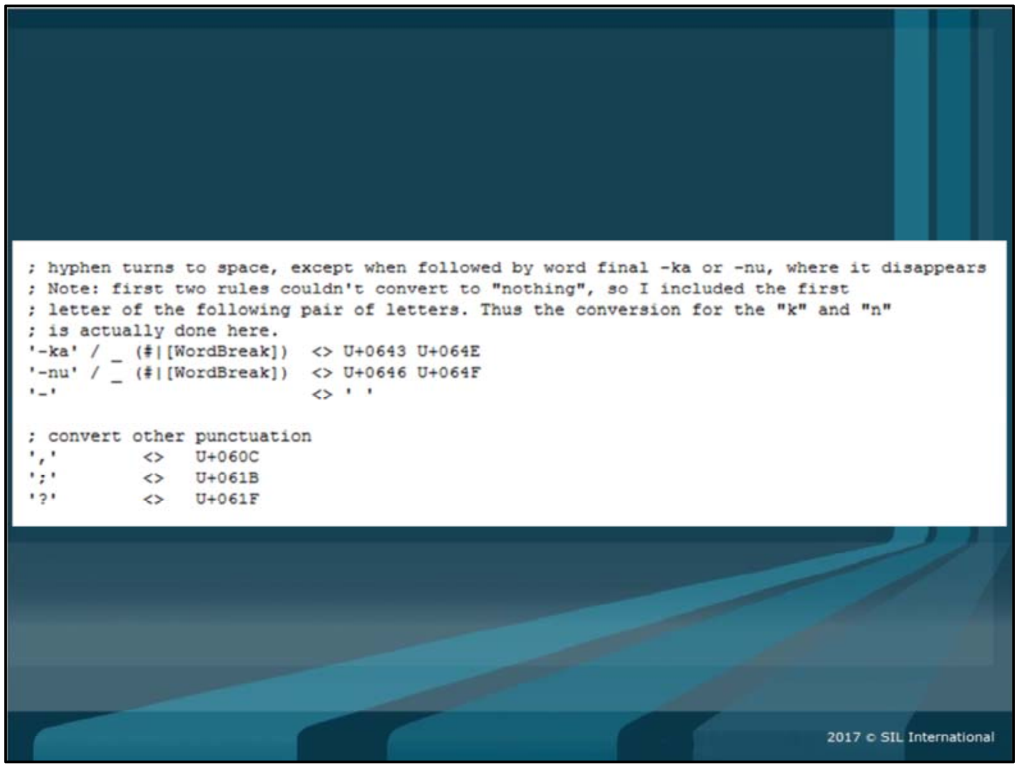
Unicode Name	U Value	Chars
latin_capital_letter_a	U+0041	A
latin_small_letter_l	U+006C	l
latin_small_letter_l	U+006C	l
latin_small_letter_a	U+0061	a
latin_small_letter_h	U+0068	h

With this change, our conversion of Allah works out the way we want it to. One nice side-effect of handling Proper Names this way is that the Round-trip is performed very easily. When we reverse the conversion, the Arabic script form just gets changed into a unique code, the rest of the passes are run backwards to change other text, and then the first pass changes those unique codes back into Roman script Proper Names.



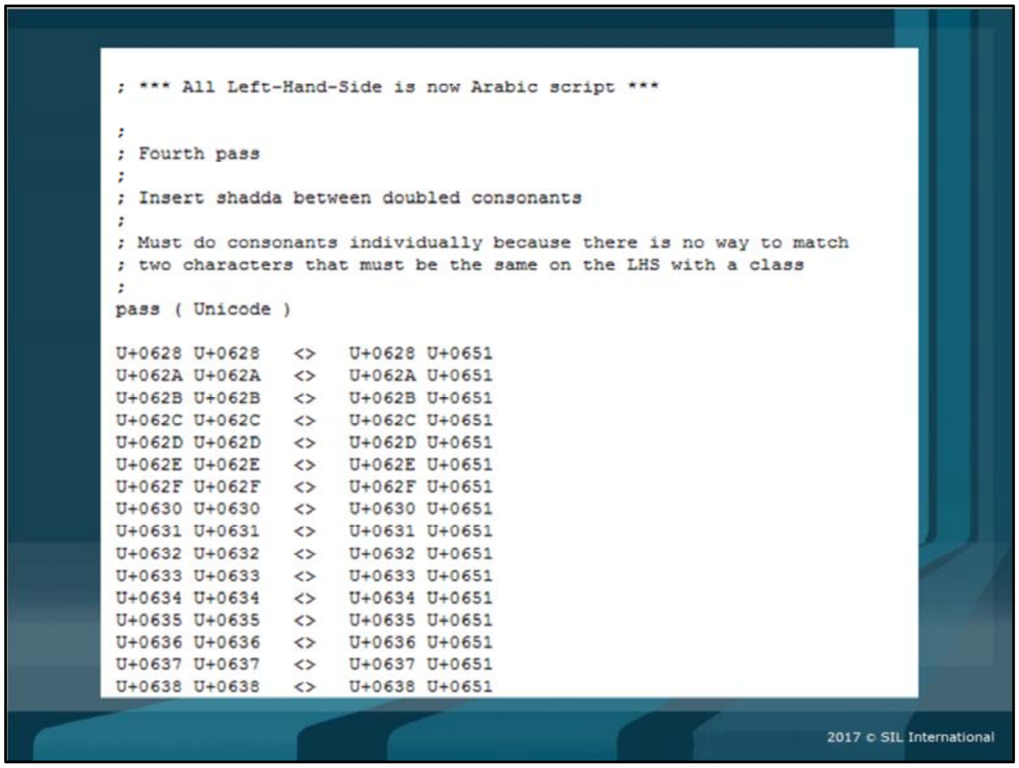
```
class[vowel] = ( 'a' 'e' 'i' 'o' 'u' )

; take care of dipthongs first: ay, ey, oy, uy, aw, ew, iw, ow
; note that dipthongs can never be followed by another vowel
; note that word initial forms are different
'ay' / (#|[WordBreak]) _ ^[vowel]  <>   U+0623 U+064E U+064A U+0652
'ay' /                 _ ^[vowel]  <>   U+064E U+064A U+0652
'ey' / (#|[WordBreak]) _ ^[vowel]  <>   U+0623 U+065A U+064A U+0652
'ey' /                 _ ^[vowel]  <>   U+065A U+064A U+0652
'oy' / (#|[WordBreak]) _ ^[vowel]  <>   U+0623 U+065B U+064A U+0652
'oy' /                 _ ^[vowel]  <>   U+065B U+064A U+0652
'uy' / (#|[WordBreak]) _ ^[vowel]  <>   U+0623 U+064F U+064A U+0652
'uy' /                 _ ^[vowel]  <>   U+064F U+064A U+0652
'aw' / (#|[WordBreak]) _ ^[vowel]  <>   U+0623 U+064E U+0648 U+0652
'aw' /                 _ ^[vowel]  <>   U+064E U+0648 U+0652
'ew' / (#|[WordBreak]) _ ^[vowel]  <>   U+0623 U+065A U+0648 U+0652
'ew' /                 _ ^[vowel]  <>   U+065A U+0648 U+0652
'iw' / (#|[WordBreak]) _ ^[vowel]  <>   U+0625 U+0650 U+0648 U+0652
'iw' /                 _ ^[vowel]  <>   U+0650 U+0648 U+0652
'ow' / (#|[WordBreak]) _ ^[vowel]  <>   U+0623 U+065B U+0648 U+0652
'ow' /                 _ ^[vowel]  <>   U+065B U+0648 U+0652
```

Another thing we need to handle in our Maba conversion is vowel diphthongs. They are handled in a similar way to that of vowels and long vowels. You can see that we include the WordBreak context in the same way to determine if the diphthong appears at the beginning of a word. But in the case of the diphthongs, we also need to make sure that there is not a vowel following the diphthong. If there is a vowel following it, then the “w” or “y” of the diphthong is actually a consonant, and we don’t want to convert it. So in the context for our diphthongs we say that the character following can NOT be a vowel. The caret symbol is a negation. So if a vowel follows the diphthong, that context won’t match and it won’t be converted.



```
; hyphen turns to space, except when followed by word final -ka or -nu, where it disappears
; Note: first two rules couldn't convert to "nothing", so I included the first
; letter of the following pair of letters. Thus the conversion for the "k" and "n"
; is actually done here.
'-ka' / _ (#|[WordBreak])  <> U+0643 U+064E
'-nu' / _ (#|[WordBreak])  <> U+0646 U+064F
'-'                        <> ' '

; convert other punctuation
','        <>   U+060C
';'        <>   U+061B
'?'        <>   U+061F
```

Here are a couple of other miscellaneous conversions that we do in the Maba mapping file. The first section takes care of a couple of very specific constructions in the Maba language. There are two suffixes in Maba that are written in Roman script with a hyphen, “-ka” and “-na”. The Arabic script version of those suffixes attaches directly to the previous word without a hyphen. Anywhere else we see a hyphen, it just gets turned into a space.
This is an example of a very specific and precise conversion that we make for this specific language.
And the last section here just changes some punctuation to their Arabic script forms.



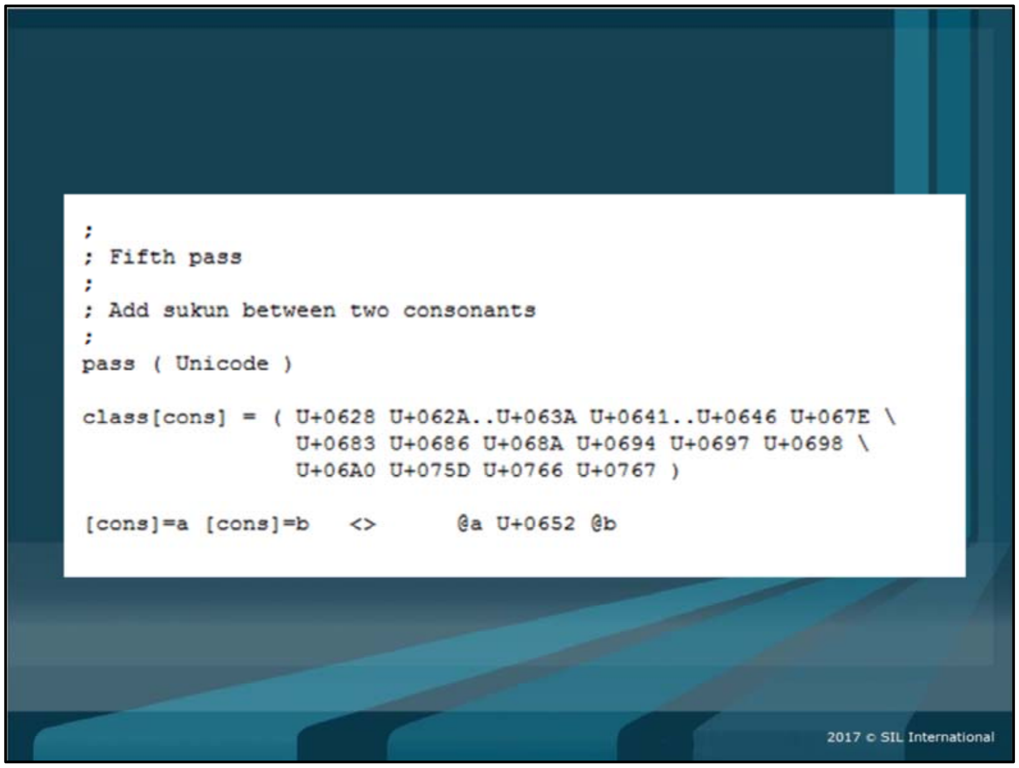
```
; *** All Left-Hand-Side is now Arabic script ***

;
; Fourth pass
;
; Insert shadda between doubled consonants
;
; Must do consonants individually because there is no way to match
; two characters that must be the same on the LHS with a class
;
pass ( Unicode )

U+0628 U+0628   <>   U+0628 U+0651
U+062A U+062A   <>   U+062A U+0651
U+062B U+062B   <>   U+062B U+0651
U+062C U+062C   <>   U+062C U+0651
U+062D U+062D   <>   U+062D U+0651
U+062E U+062E   <>   U+062E U+0651
U+062F U+062F   <>   U+062F U+0651
U+0630 U+0630   <>   U+0630 U+0651
U+0631 U+0631   <>   U+0631 U+0651
U+0632 U+0632   <>   U+0632 U+0651
U+0633 U+0633   <>   U+0633 U+0651
U+0634 U+0634   <>   U+0634 U+0651
U+0635 U+0635   <>   U+0635 U+0651
U+0636 U+0636   <>   U+0636 U+0651
U+0637 U+0637   <>   U+0637 U+0651
U+0638 U+0638   <>   U+0638 U+0651
```

2017 © SIL International

I have a couple more passes that I would like to show you, but I want to draw your attention first to the comment that appears before this pass. The pass right before this one was the main character conversion pass. In that pass, everything that was still in Roman script should have been converted to Arabic script. That means that any conversions from here on out will just change Arabic script to Arabic script. What was a pass we saw before that changed Roman script to Roman script? Right, the conversion to lowercase.

So generally the overall design will probably have a pass for marking proper names, possibly some Roman script conversion passes, then one main pass to do the bulk of the conversion from Roman to Arabic, then addition passes for making adjustments in the Arabic script, and a final pass for changing our proper name markers into their final Arabic script form.

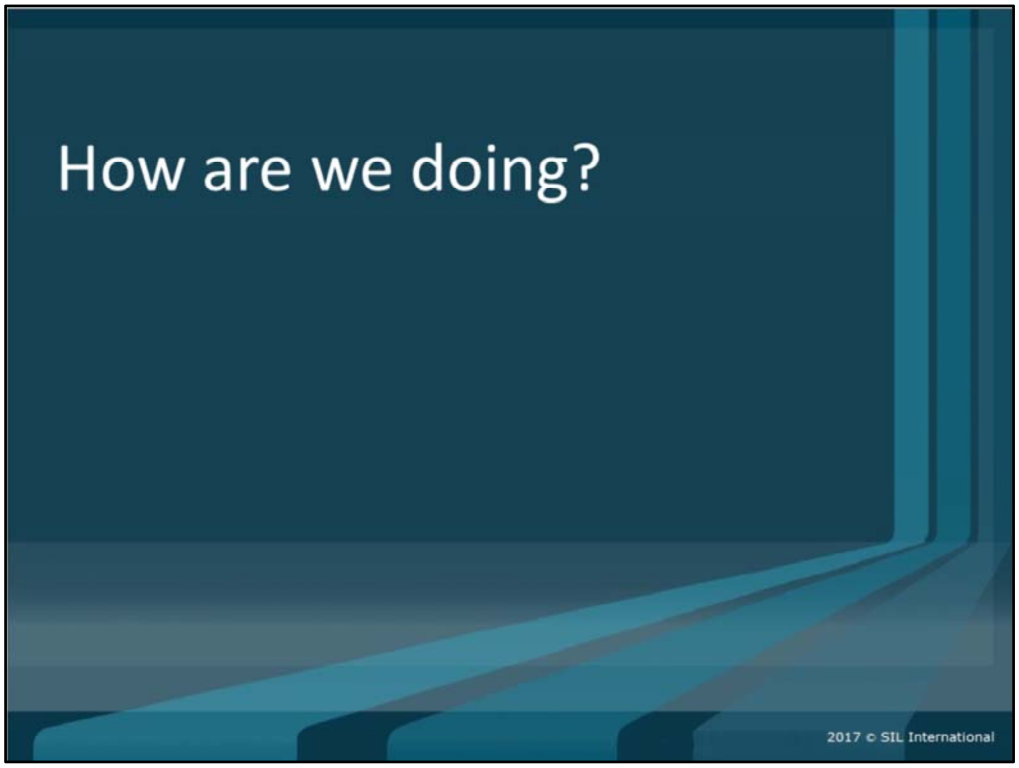
So this pass here is one of the latter ones for adjusting the Arabic script. I mentioned that one problem running Allah through the normal conversion was that a double LAM was being changed into a LAM shadda. This is the section of the mapping file that does that. It simply looks for specific doubled consonants and changes them into that consonant followed by a shadda. Remember that this pass is run after we have already attempted to convert everything into Arabic script, so the left-hand-side character matches are in the Arabic 0600 block.



```
;
; Fifth pass
;
; Add sukun between two consonants
;
pass ( Unicode )

class[cons] = ( U+0628 U+062A..U+063A U+0641..U+0646 U+067E \
                U+0683 U+0686 U+068A U+0694 U+0697 U+0698 \
                U+06A0 U+075D U+0766 U+0767 )

[cons]=a [cons]=b   <>      @a U+0652 @b
```

2017 © SIL International

And one last thing we do is look for places where there are two consonants next to each other and put a sukun between them. Have you ever noticed that Arabic readers always stick a vowel sound between two consonants, even if there isn't one there. The sukun is intended to mark a place where there specifically is not supposed to be a vowel. Some languages may find that marking sukuns is helpful, and this section of the mapping file can do that.



How are we doing?

2017 © SIL International

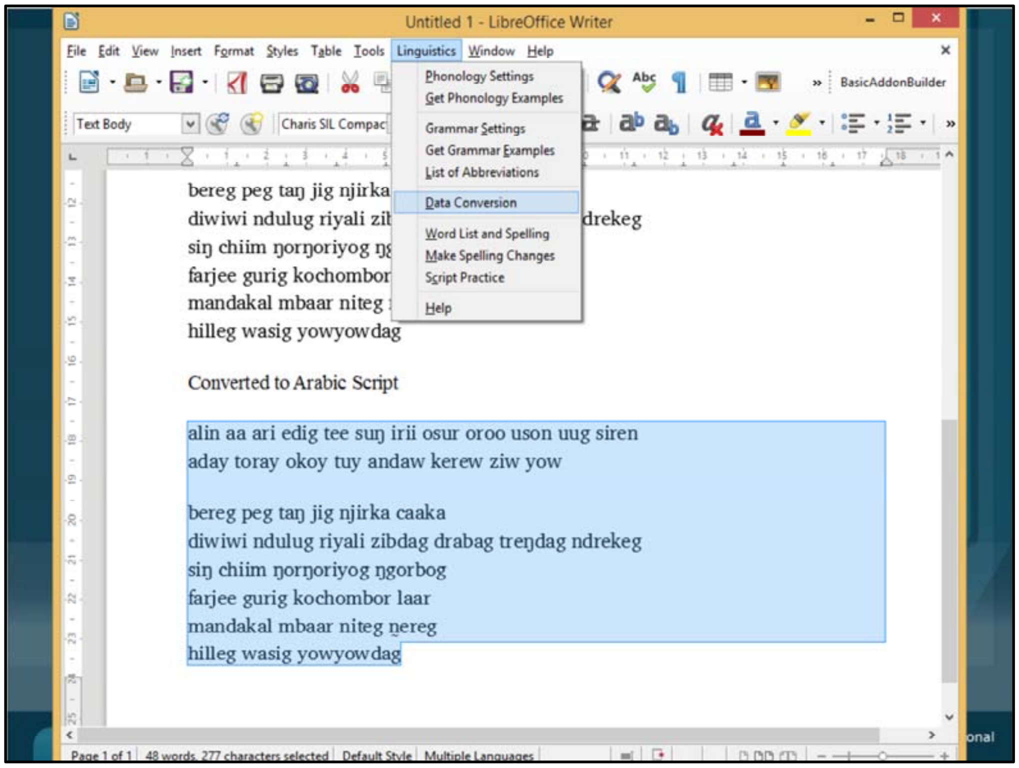
Time for another process check.
What will generally be the organization of passes in our mapping file?
Tag proper names, RS passes, main conversion pass, AS passes, insert AS proper names
What didn’t you follow, or what questions do you have?

Installing a New Converter

- From Data Conversion macro

2017 © SIL International

It’s great that you can use the sample boxes in the TECkit Mapping Editor to perform a quick conversion with your mapping file, but at some point you’re going to want to do “real” conversions, converting large sections of text in LibreOffice or Word, or converting translations like we saw in Paratext. To do that, you need to install your mapping file as a converter in SIL Converters, which is generally done by using the Data Conversion macro that we saw before.


Untitled 1 - LibreOffice Writer

bereg peg taŋ jig njirka
diwiwi ndulug riyali zib drekeg
siŋ chiim ŋorŋoriyog ŋg
farjee gurig kochombor
mandakal mbaar niteg
hilleg wasig yowyowdag

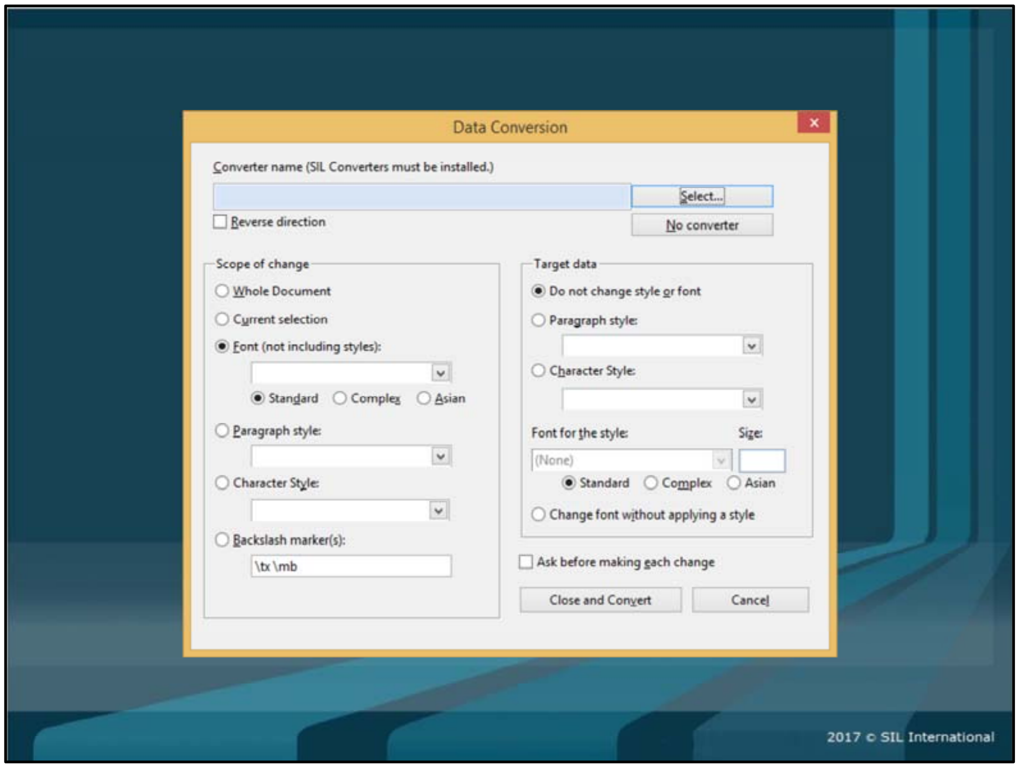
Converted to Arabic Script

alin aa ari edig tee suŋ irii osur oroo uson uug siren
aday toray okoy tuy andaw kerew ziw yow

bereg peg taŋ jig njirka caaka
diwiwi ndulug riyali zibdag drabag treŋdag ndrekeg
siŋ chiim ŋorŋoriyog ŋgorbog
farjee gurig kochombor laar
mandakal mbaar niteg ŋereg
hilleg wasig yowyowdag

Page 1 of 1 | 48 words, 277 characters selected | Default Style | Multiple Languages

When I first showed you a conversion in LibreOffice, we selected Data Conversion from the Linguistics menu.



Data Conversion

Converter name (SIL Converters must be installed.)

Select... | No converter

☐ Reverse direction

Scope of change
- ○ Whole Document
- ○ Current selection
- ● Font (not including styles):
  - ● Standard ○ Complex ○ Asian
- ○ Paragraph style:
- ○ Character Style:
- ○ Backslash marker(s): \tx \mb

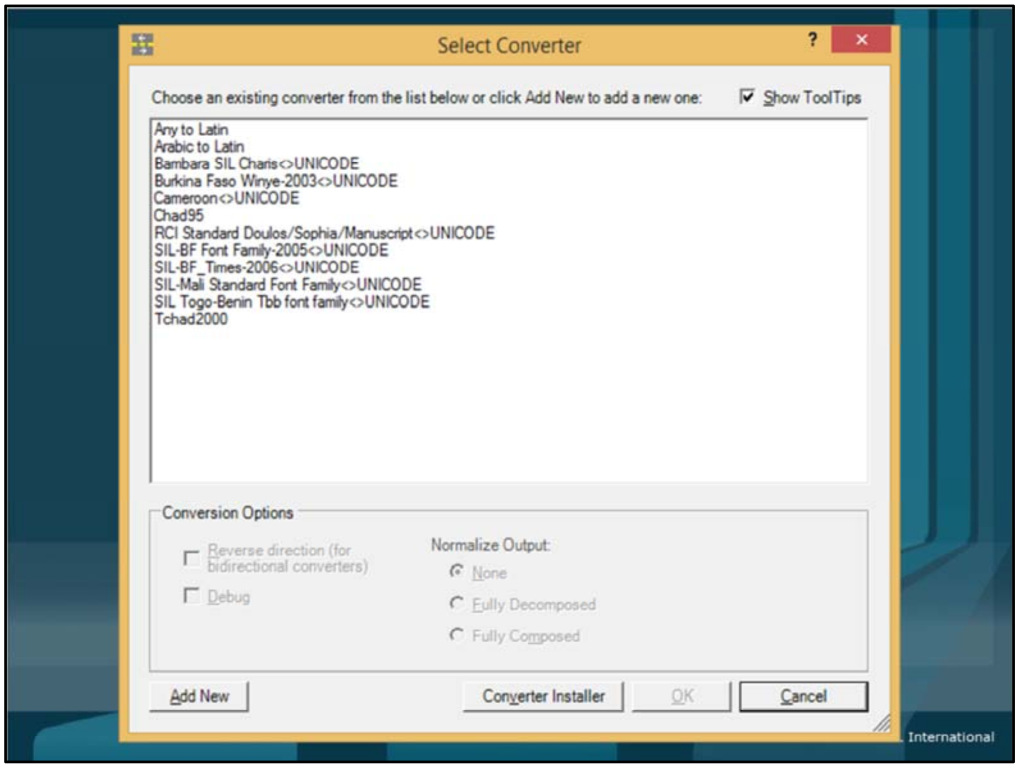
Target data
- ● Do not change style or font
- ○ Paragraph style:
- ○ Character Style:
- Font for the style: (None) Size:
  - ● Standard ○ Complex ○ Asian
- ○ Change font without applying a style

☐ Ask before making each change

Close and Convert | Cancel

2017 © SIL International

To the right of the first field here, we click Select button to select the converter to use.



Select Converter

Choose an existing converter from the list below or click Add New to add a new one: [✓ Show ToolTips]

Any to Latin
Arabic to Latin
Bambara SIL Charis<>UNICODE
Burkina Faso Winye-2003<>UNICODE
Cameroon<>UNICODE
Chad95
RCI Standard Doulos/Sophia/Manuscript<>UNICODE
SIL-BF Font Family-2005<>UNICODE
SIL-BF_Times-2006<>UNICODE
SIL-Mali Standard Font Family<>UNICODE
SIL Togo-Benin Tbb font family<>UNICODE
Tchad2000

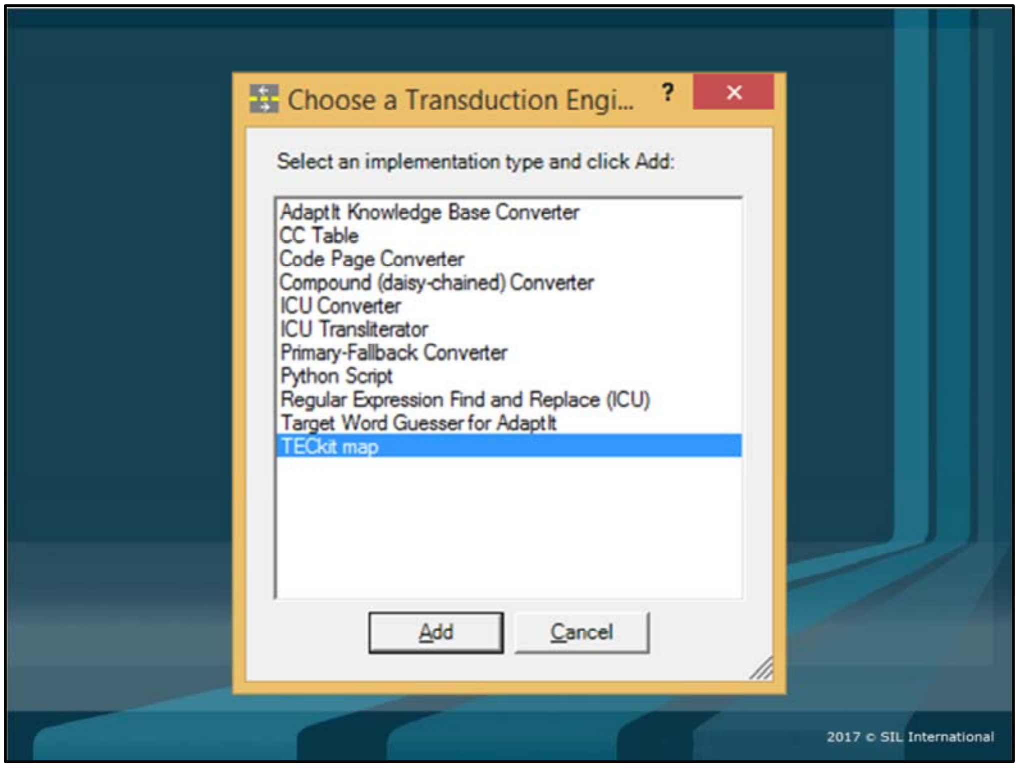
Conversion Options

Reverse direction (for bidirectional converters)

Debug

Normalize Output: [✓ None] Fully Decomposed Fully Composed

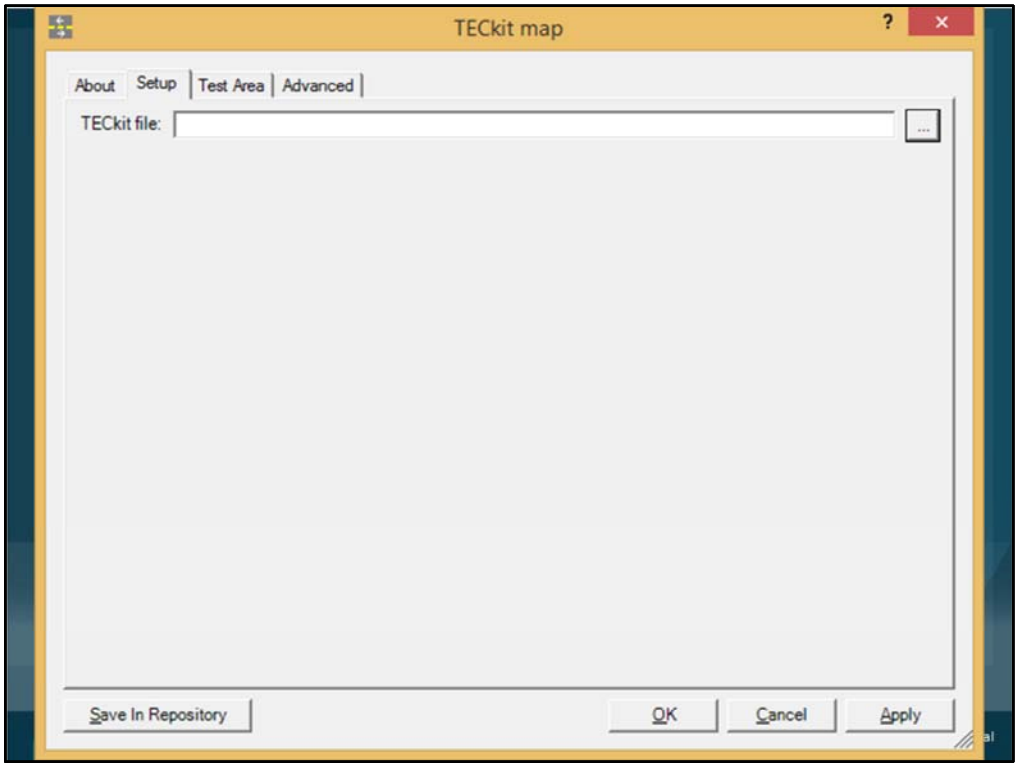
Let’s say we just finished creating our MabaAS mapping file. You can see that MabaAS is not in the list of converters. So we click on the Add New button.



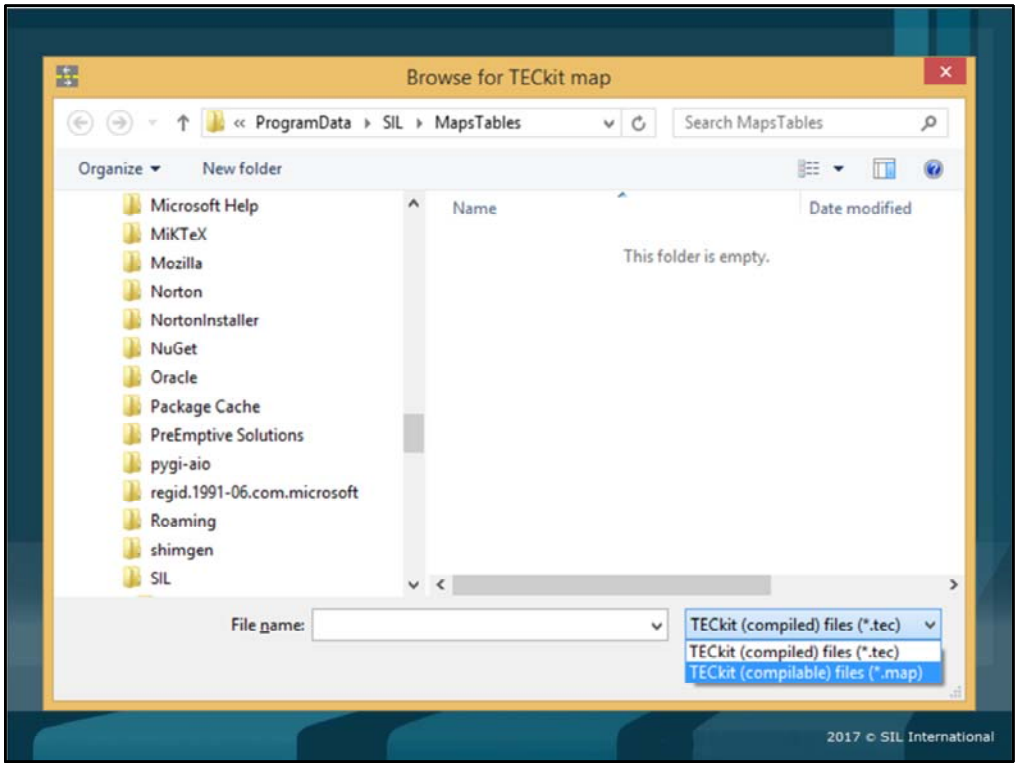
Select an implementation type and click Add:

- AdaptIt Knowledge Base Converter
- CC Table
- Code Page Converter
- Compound (daisy-chained) Converter
- ICU Converter
- ICU Transliterator
- Primary-Fallback Converter
- Python Script
- Regular Expression Find and Replace (ICU)
- Target Word Guesser for AdaptIt
- TECkit map

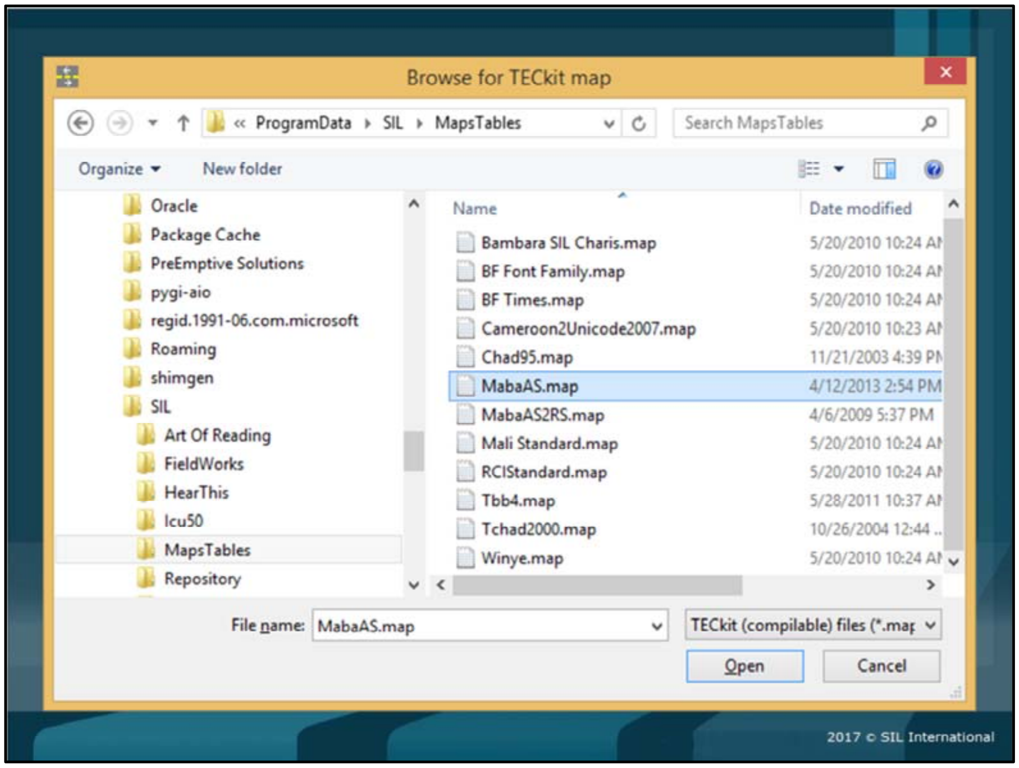
There are a number of different types of converters that can be used. Click on TECkit map at the bottom and click the Add button.



In the window that appears, click on the Setup tab, then on the three dots to browse for your mapping file.



I would recommend that you put a copy of your mapping file in C:\ProgramData\SIL\MapsTables, and use that copy in SIL Converters. Then you will still be able to use the converter even if you move your original file around or rename it. The default file type is compiled TECkit files, with a .tec extension, so you need to switch to viewing the .map text files.

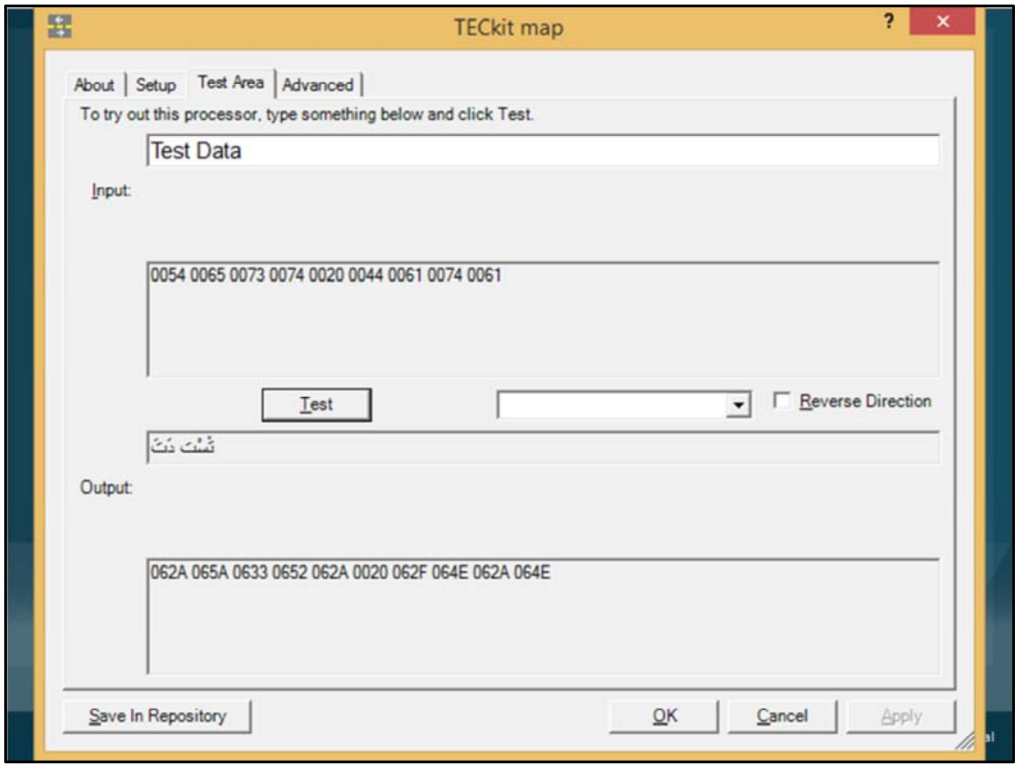


Name	Date modified
Bambara SIL Charis.map	5/20/2010 10:24 AM
BF Font Family.map	5/20/2010 10:24 AM
BF Times.map	5/20/2010 10:24 AM
Cameroon2Unicode2007.map	5/20/2010 10:23 AM
Chad95.map	11/21/2003 4:39 PM
MabaAS.map	4/12/2013 2:54 PM
MabaAS2RS.map	4/6/2009 5:37 PM
Mali Standard.map	5/20/2010 10:24 AM
RCIStandard.map	5/20/2010 10:24 AM
Tbb4.map	5/28/2011 10:37 AM
Tchad2000.map	10/26/2004 12:44 ...
Winye.map	5/20/2010 10:24 AM

File name: MabaAS.map

TECkit (compilable) files (*.map

We select MabaAS.map and open it.



To try out this processor, type something below and click Test.

Input:

```
Test Data
```

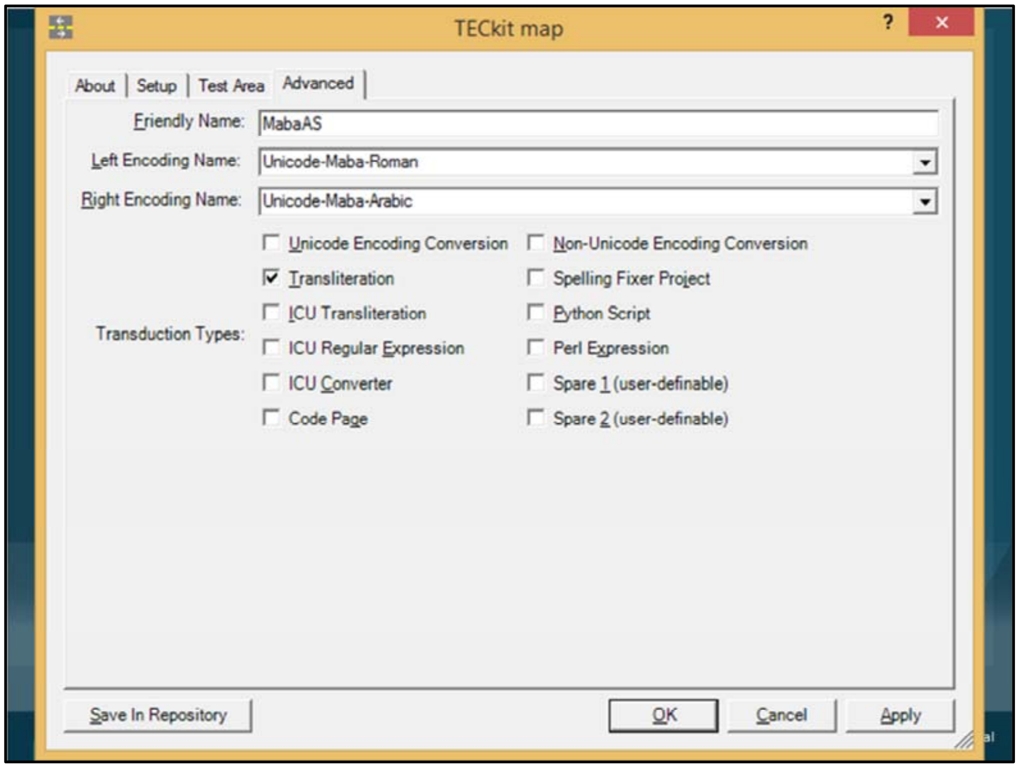
```
0054 0065 0073 0074 0020 0044 0061 0074 0061
```

Output:

تٚسْت دَتَ

```
062A 065A 0633 0652 062A 0020 062F 064E 062A 064E
```

You can use the Test Area tab if you like to make sure that the mapping file is working. It is similar to the sample boxes in the editor. It also shows the Unicode input and output codes.


TECkit map

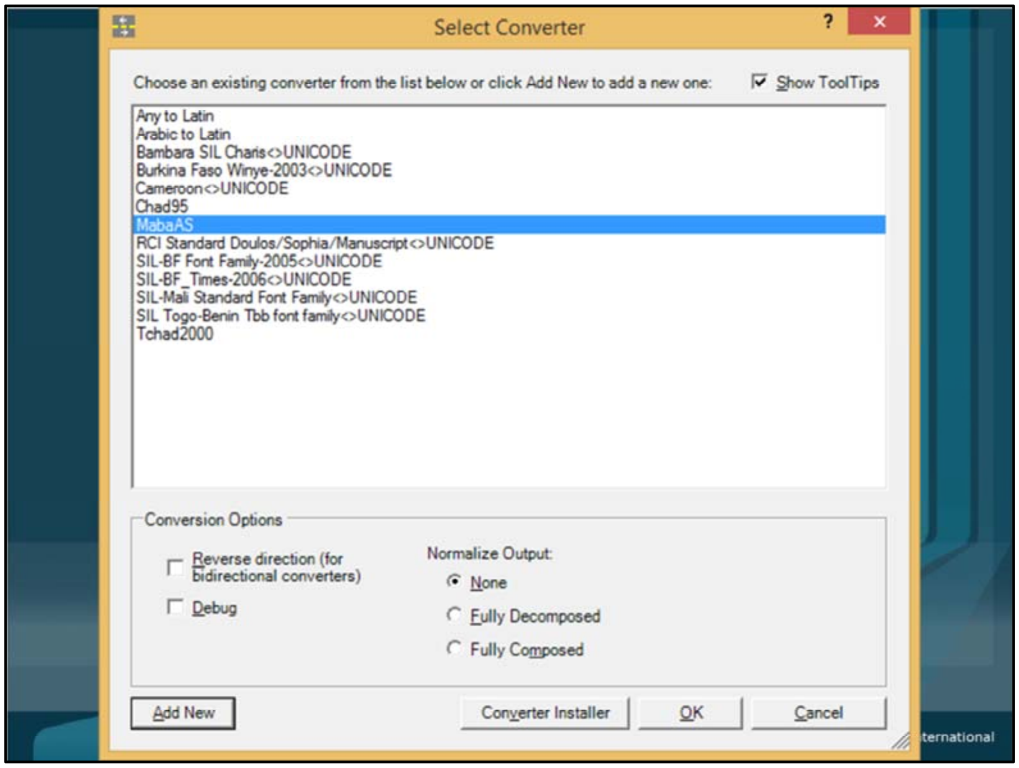
Friendly Name: MabaAS

Left Encoding Name: Unicode-Maba-Roman

Right Encoding Name: Unicode-Maba-Arabic

Transduction Types:
- [ ] Unicode Encoding Conversion
- [ ] Non-Unicode Encoding Conversion
- [✓] Transliteration
- [ ] Spelling Fixer Project
- [ ] ICU Transliteration
- [ ] Python Script
- [ ] ICU Regular Expression
- [ ] Perl Expression
- [ ] ICU Converter
- [ ] Spare 1 (user-definable)
- [ ] Code Page
- [ ] Spare 2 (user-definable)

And then on the Advanced tab, you can fill in the information about the mapping file. You should check transliteration, since this is a Unicode script conversion. Then click the Save in Repository button, and then click OK.



Select Converter

Choose an existing converter from the list below or click Add New to add a new one: [✓ Show ToolTips]

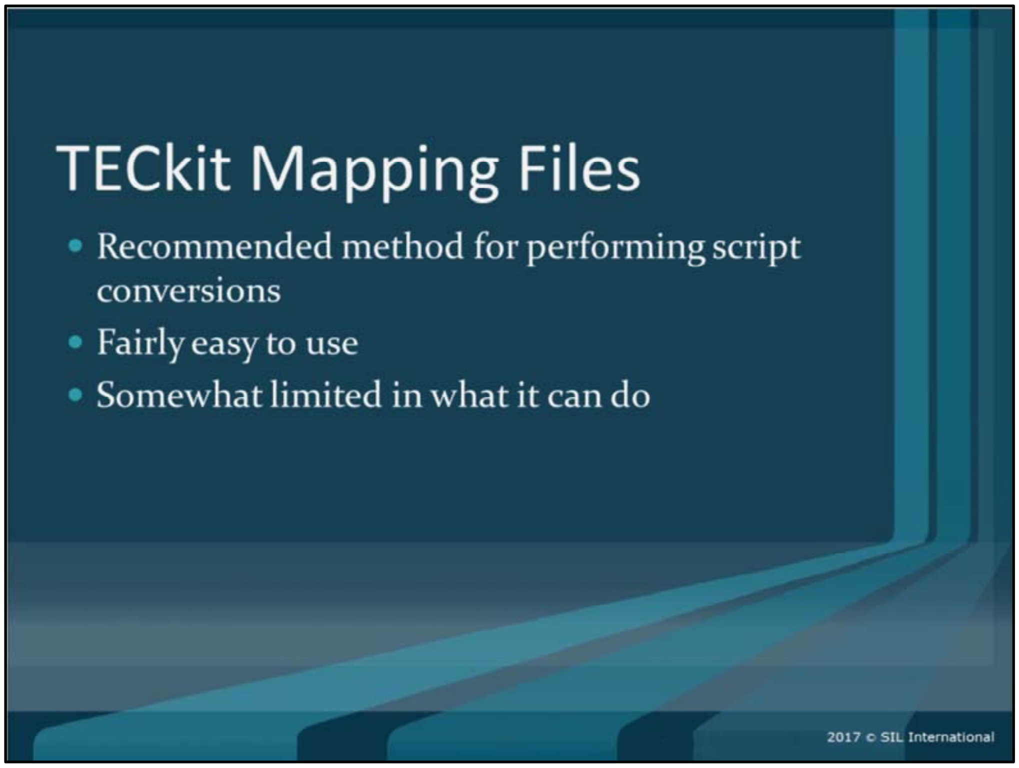
Any to Latin
Arabic to Latin
Bambara SIL Charis<>UNICODE
Burkina Faso Winye-2003<>UNICODE
Cameroon<>UNICODE
Chad95
MabaAS
RCI Standard Doulos/Sophia/Manuscript<>UNICODE
SIL-BF Font Family-2005<>UNICODE
SIL-BF_Times-2006<>UNICODE
SIL-Mali Standard Font Family<>UNICODE
SIL Togo-Benin Tbb font family<>UNICODE
Tchad2000

Conversion Options

Reverse direction (for bidirectional converters)
Debug

Normalize Output:
[✓ None]
Fully Decomposed
Fully Composed

Now we have that new converter in the repository, so we can select it and click OK, to return to the Data Conversion macro with our new converter selected. It’s also possible to add a converter to the repository directly from the editor. But it’s helpful to know how to do it manually as well, since you may have to install converters for people who aren’t using the editor.

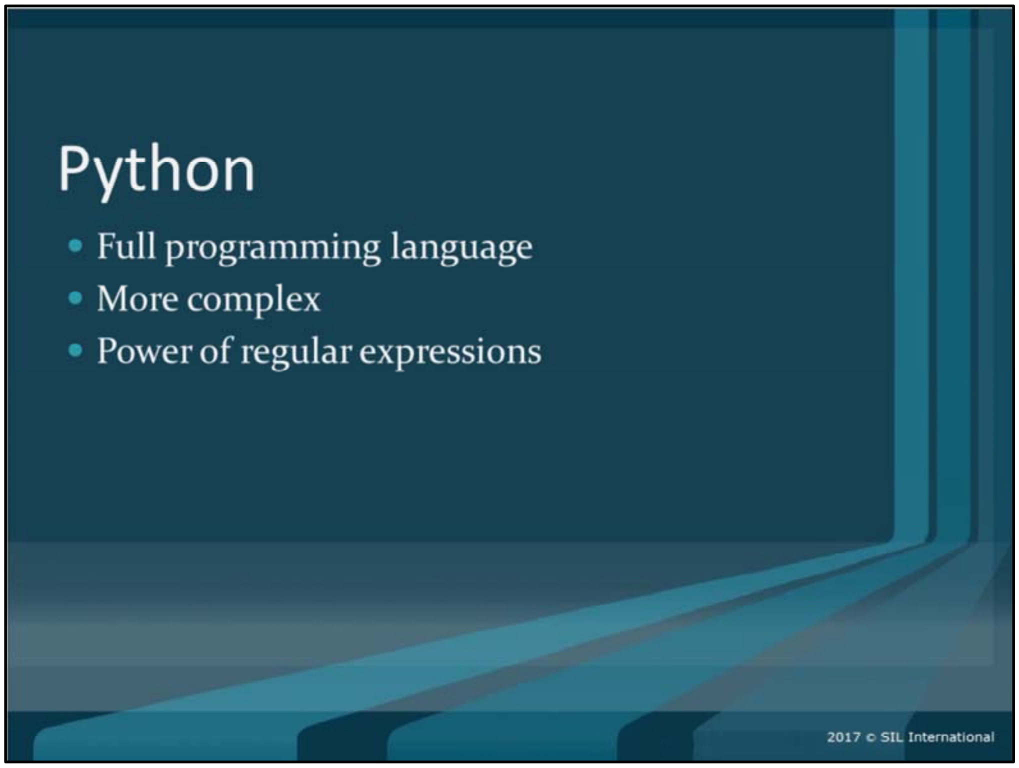


TECkit Mapping Files

- Recommended method for performing script conversions
- Fairly easy to use
- Somewhat limited in what it can do

2017 © SIL International

So that's all I wanted to share with you about creating TECkit mapping files and using them to perform script conversions. This is the main technique that we are encouraging you to use for your script conversions, as we help you with those this week.
I would like to briefly show you a couple of other techniques, but before we move on, are there any questions?
I have one question for you. What did you think about the TECkit programming language?
□ It's fairly easy to do simple things with it.
□ But it's also limited in what it can do. For example, inserting the shaddas required writing a separate rule for every consonant. Not the end of the world, but that shows that it has some limitations.

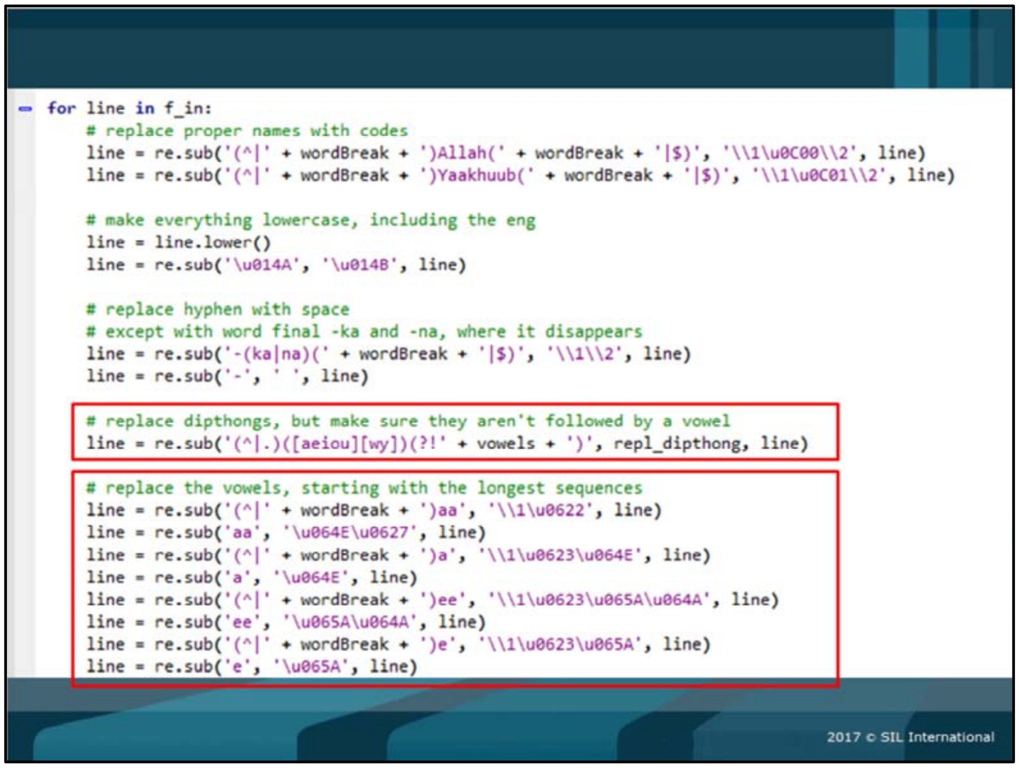


Python

- Full programming language
- More complex
- Power of regular expressions

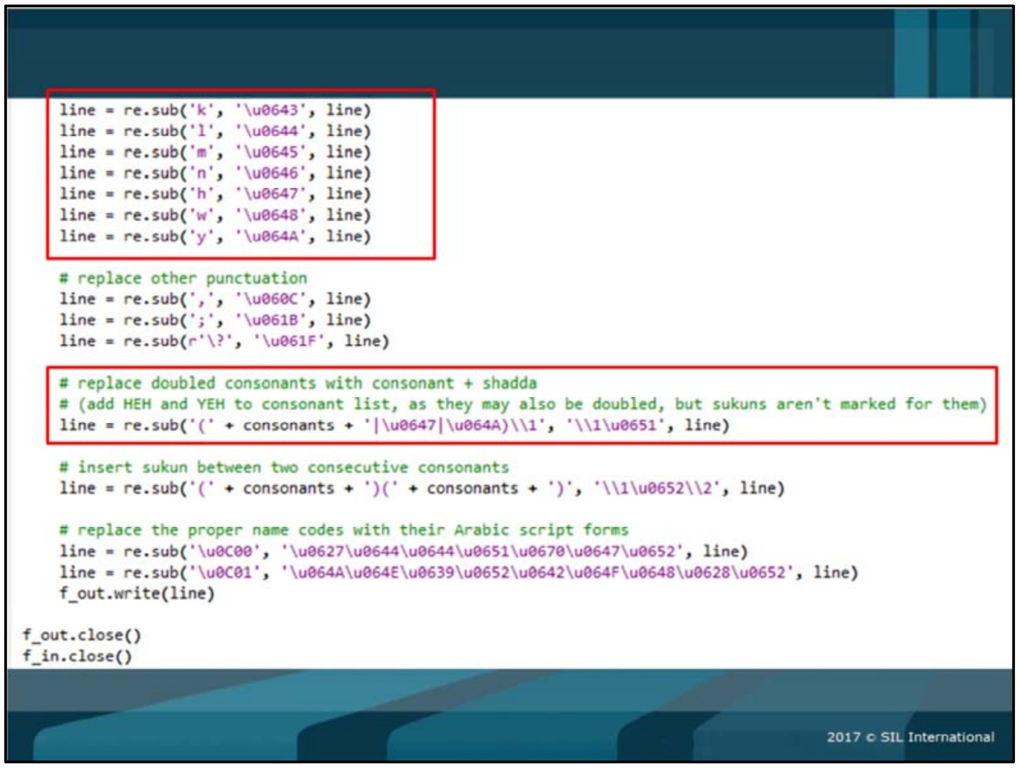
2017 © SIL International

Python, however, is a full-blown programming language, and you can pretty much do any conversion task you want with it.
□ But it's also a lot more complex. You can do some simple tasks fairly easily, but it really takes a lifetime to master.
□ One nice thing about Python is that it allows the use of regular expressions – a very powerful way of matching and replacing text, which is exactly the kind of thing we want to do.



```python
for line in f_in:
    # replace proper names with codes
    line = re.sub('(^|' + wordBreak + ')Allah(' + wordBreak + '|$)', '\\1\u0C00\\2', line)
    line = re.sub('(^|' + wordBreak + ')Yaakhuub(' + wordBreak + '|$)', '\\1\u0C01\\2', line)

    # make everything lowercase, including the eng
    line = line.lower()
    line = re.sub('\u014A', '\u014B', line)

    # replace hyphen with space
    # except with word final -ka and -na, where it disappears
    line = re.sub('-(ka|na)(' + wordBreak + '|$)', '\\1\\2', line)
    line = re.sub('-', ' ', line)

    # replace dipthongs, but make sure they aren't followed by a vowel
    line = re.sub('(^|.)([aeiou][wy])(?!' + vowels + ')', repl_dipthong, line)

    # replace the vowels, starting with the longest sequences
    line = re.sub('(^|' + wordBreak + ')aa', '\\1\u0622', line)
    line = re.sub('aa', '\u064E\u0627', line)
    line = re.sub('(^|' + wordBreak + ')a', '\\1\u0623\u064E', line)
    line = re.sub('a', '\u064E', line)
    line = re.sub('(^|' + wordBreak + ')ee', '\\1\u0623\u065A\u064A', line)
    line = re.sub('ee', '\u065A\u064A', line)
    line = re.sub('(^|' + wordBreak + ')e', '\\1\u0623\u065A', line)
    line = re.sub('e', '\u065A', line)
```

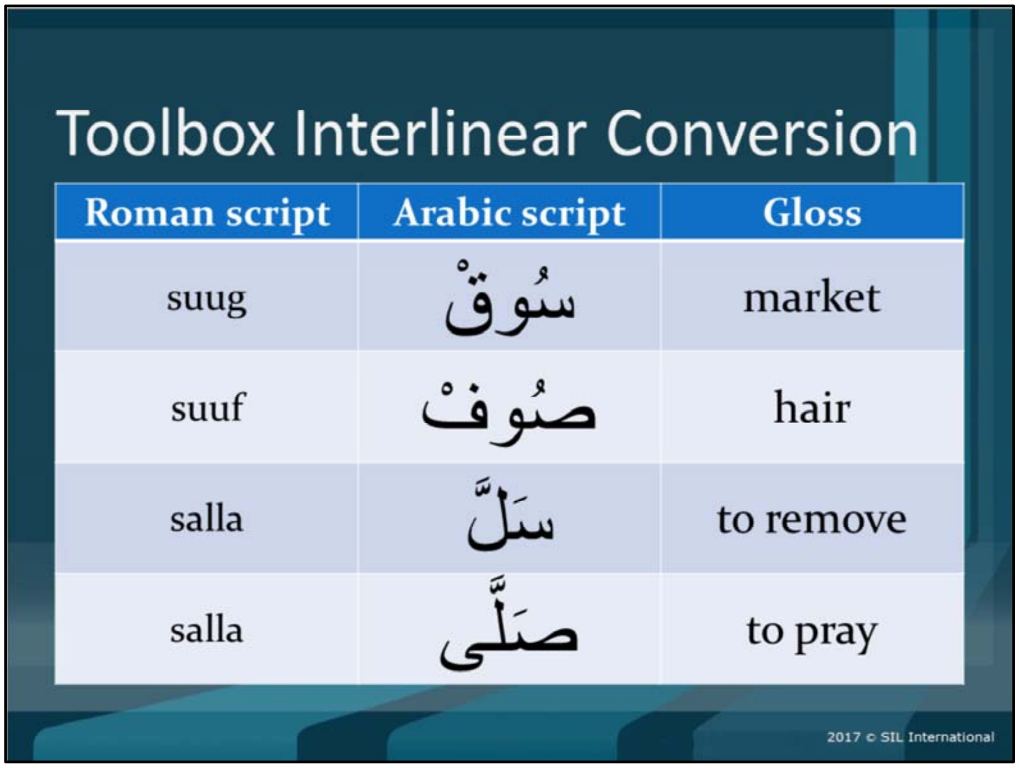
To give you a sense of what is involved in writing a conversion as a Python script, I rewrote our MabaAS mapping file in Python. There are different ways to do it, but I chose to read text from an input file one line at a time, convert that line, and then write the converted line out to an output file. The conversion process flows in a similar way to our mapping file. You can see here in the green comments that we deal with proper names, make everything lowercase, etc. One thing we have to be more intentional about is that Python just runs the commands in the order they appear in the program, so we have to make sure that we replace the longer strings before the individual letters. That was one nice thing about our mapping file – it always matched the longest string first, so we didn't have to worry too much about what order they were in.
□ But this code shows one advantage of using Python. This is the code for converting the diphthongs. In the mapping file it was 16 lines, but here we are able to use regular expressions to reduce the matching of the diphthongs down to one line. The sets [aeiou] and [wy] in the middle allow us to easily match all of the possible diphthongs. The replacement is handled by a function which handles word initial diphthongs and makes sure the diphthong is valid. One could argue that this code is easier to understand than a long list of matching rules, where you might not be able to see the patterns.
□ And if you find that too obscure, you can just write rules that are very similar to the way the rules are written in the mapping file. That's what I did for the vowel rules here. There are 4 rules for each vowel, whether or not the vowel is at the beginning of a word, and whether it is long or short, exactly like our mapping file. So with Python you can write simpler code in some places, but where you need more control, you can take advantage of the depth of the language to do what you need to do.



```
    line = re.sub('k', '\u0643', line)
    line = re.sub('l', '\u0644', line)
    line = re.sub('m', '\u0645', line)
    line = re.sub('n', '\u0646', line)
    line = re.sub('h', '\u0647', line)
    line = re.sub('w', '\u0648', line)
    line = re.sub('y', '\u064A', line)

    # replace other punctuation
    line = re.sub(',', '\u060C', line)
    line = re.sub(';', '\u061B', line)
    line = re.sub(r'\?', '\u061F', line)

    # replace doubled consonants with consonant + shadda
    # (add HEH and YEH to consonant list, as they may also be doubled, but sukuns aren't marked for them)
    line = re.sub('(' + consonants + '|\u0647|\u064A)\\1', '\\1\u0651', line)

    # insert sukun between two consecutive consonants
    line = re.sub('(' + consonants + ')(' + consonants + ')', '\\1\u0652\\2', line)

    # replace the proper name codes with their Arabic script forms
    line = re.sub('\u0C00', '\u0627\u0644\u0644\u0651\u0670\u0647\u0652', line)
    line = re.sub('\u0C01', '\u064A\u064E\u0639\u0652\u0642\u064F\u0648\u0628\u0652', line)
    f_out.write(line)

f_out.close()
f_in.close()
```

2017 © SIL International

This is later on in the Python script.
□ You can see the very simple replacements of the consonants.
□ And here is the replacement of doubled consonants with a consonant and a shadda. In the mapping file it took us 37 lines to cover all of the possible consonants. In Python, the consonants are defined by a couple of lines at the top, and the conversion of doubled consonants to consonant plus shadda takes only one line of code.
□ So there are some advantages to using Python, but it's also a pretty steep learning curve. So if a mapping file can do what you need for script conversion, that's definitely the way to go.

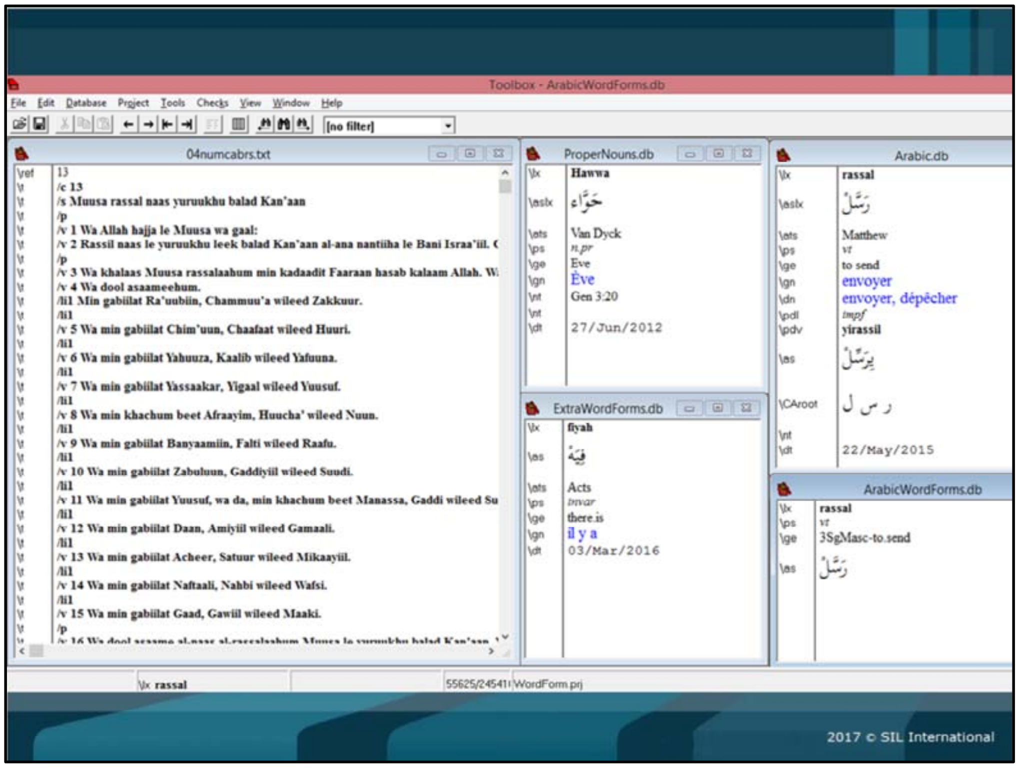


Toolbox Interlinear Conversion

Roman script	Arabic script	Gloss
suug	سُوقْ	market
suuf	صُوفْ	hair
salla	سَلَّ	to remove
salla	صَلَّى	to pray

2017 © SIL International

And I just wanted to talk about one more script conversion possibility. In Chadian Arabic, the Roman script is written for non-mother tongue speakers, who can’t distinguish between a “seen” and a “sad”. But those distinctions are made in many cases in the Arabic script. So when you convert an “s” from Roman script, do you use a “seen” or a “sad”? How can you tell? Is there a context that we can base our decision on? Yes… but the context is the whole word!


Toolbox - ArabicWordForms.db
File Edit Database Project Tools Checks View Window Help
[no filter]
04numcabrs.txt
ProperNouns.db
ExtraWordForms.db
Arabic.db
ArabicWordForms.db

```
\ref 13
\t   /c 13
\t   /s Muusa rassal naas yuruukhu balad Kan'aan
\t   /p
\t   /v 1 Wa Allah hajja le Muusa wa gaal:
\t   /v 2 Rassil naas le yuruukhu leek balad Kan'aan al-ana nantiiha le Bani Israa'iil. (
\t   /p
\t   /v 3 Wa khalaas Muusa rassalaahum min kadaadit Faaraan hasab kalaam Allah. W
\t   /v 4 Wa dool asaameehum.
\t   /li1 Min gabiilat Ra'uubiin, Chammuu'a wileed Zakkuur.
\t   /li1
\t   /v 5 Wa min gabiilat Chim'uun, Chaafaat wileed Huuri.
\t   /li1
\t   /v 6 Wa min gabiilat Yahuuza, Kaalib wileed Yafuuna.
\t   /li1
\t   /v 7 Wa min gabiilat Yassaakar, Yigaal wileed Yuusuf.
\t   /li1
\t   /v 8 Wa min khachum beet Afraayim, Huucha' wileed Nuun.
\t   /li1
\t   /v 9 Wa min gabiilat Banyaamiin, Falti wileed Raafu.
\t   /li1
\t   /v 10 Wa min gabiilat Zabuluun, Gaddiyiil wileed Suudi.
\t   /li1
\t   /v 11 Wa min gabiilat Yuusuf, wa da, min khachum beet Manassa, Gaddi wileed Su
\t   /li1
\t   /v 12 Wa min gabiilat Daan, Amiyiil wileed Gamaali.
\t   /li1
\t   /v 13 Wa min gabiilat Acheer, Satuur wileed Mikaayiil.
\t   /li1
\t   /v 14 Wa min gabiilat Naftaali, Nahbi wileed Wafsi.
\t   /li1
\t   /v 15 Wa min gabiilat Gaad, Gawiil wileed Maaki.
\t   /p
```

```
\lx     Hawwa
\aslx   حَوَّاء
\ats    Van Dyck
\ps     n.pr
\ge     Eve
\gn     Ève
\nt     Gen 3:20
\nt
\dt     27/Jun/2012
```

```
\lx     fiyah
\as     فِيَهْ
\ats    Acts
\ps     invar
\ge     there.is
\gn     il y a
\dt     03/Mar/2016
```

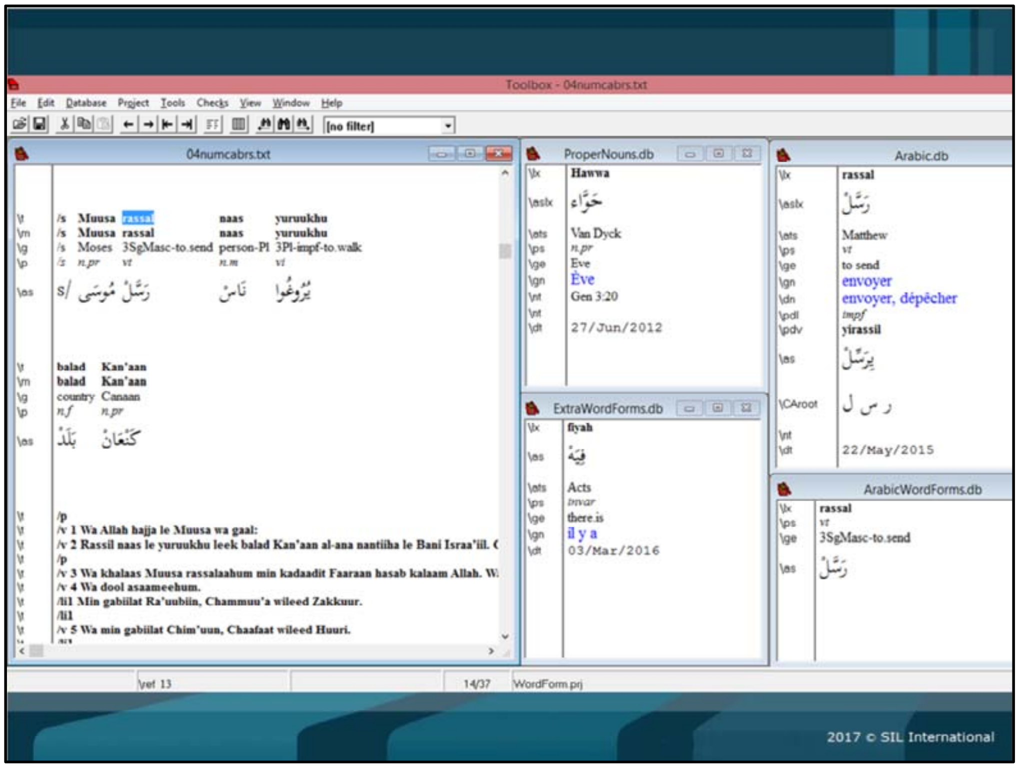
```
\lx     rassal
\aslx   رَسَّلْ
\ats    Matthew
\ps     vt
\ge     to send
\gn     envoyer
\dn     envoyer, dépêcher
\pdl    impf
\pdv    yirassil
\as     يِرَسِّلْ
\CAroot ر س ل
\nt
\dt     22/May/2015
```

```
\lx     rassal
\ps     vt
\ge     3SgMasc-to.send
\as     رَسَّلْ
```

```
\lx rassal     55625/245411 WordForm.prj
```

2017 © SIL International

So let's assume for a minute that we have a magical database called Arabic Word Forms, and that it has all of the different possible forms of words in both Roman and Arabic scripts, like this window in the lower right. If we had such a database, Toolbox is pretty handy and fast at basic interlinearization, so it could quickly look up words and produce an interlinear for a text. So if, for example, I put my cursor in the text at the beginning of the line that starts with Muusa, then click the Interlinearize button…



Toolbox - 04numcabrs.txt

04numcabrs.txt

```
\t    /s  Muusa  rassal           naas       yuruukhu
\m    /s  Muusa  rassal           naas       yuruukhu
\g    /s  Moses  3SgMasc-to.send  person-Pl  3Pl-impf-to.walk
\p    /s  n.pr   vt               n.m        vi
\as   s/  مُوسَى  رَسَّلْ            نَاسْ       يُرُوخُوا

\t    balad    Kan'aan
\m    balad    Kan'aan
\g    country  Canaan
\p    n.f      n.pr
\as   بَلَدْ     كَنْعَانْ

\t    /p
\t    /v 1 Wa Allah hajja le Muusa wa gaal:
\t    /v 2 Rassil naas le yuruukhu leek balad Kan'aan al-ana nantiiha le Bani Israa'iil. C
\t    /p
\t    /v 3 Wa khalaas Muusa rassalaahum min kadaadit Faaraan hasab kalaam Allah. W
\t    /v 4 Wa dool asaameehum.
\t    /li1 Min gabiilat Ra'uubiin, Chammuu'a wileed Zakkuur.
\t    /li1
\t    /v 5 Wa min gabiilat Chim'uun, Chaafaat wileed Huuri.
```

ProperNouns.db

```
\lx    Hawwa
\aslx  حَوَّاء
\ats   Van Dyck
\ps    n.pr
\ge    Eve
\gn    Ève
\nt    Gen 3:20
\nt
\dt    27/Jun/2012
```

ExtraWordForms.db

```
\lx    fiyah
\as    فِيَهْ
\ats   Acts
\ps    invar
\ge    there.is
\gn    il y a
\dt    03/Mar/2016
```

Arabic.db

```
\lx      rassal
\aslx    رَسَّلْ
\ats     Matthew
\ps      vt
\ge      to send
\gn      envoyer
\dn      envoyer, dépêcher
\pdl     impf
\pdv     yirassil
\as      يِرَسِّلْ
\CAroot  ر س ل
\nt
\dt      22/May/2015
```

ArabicWordForms.db

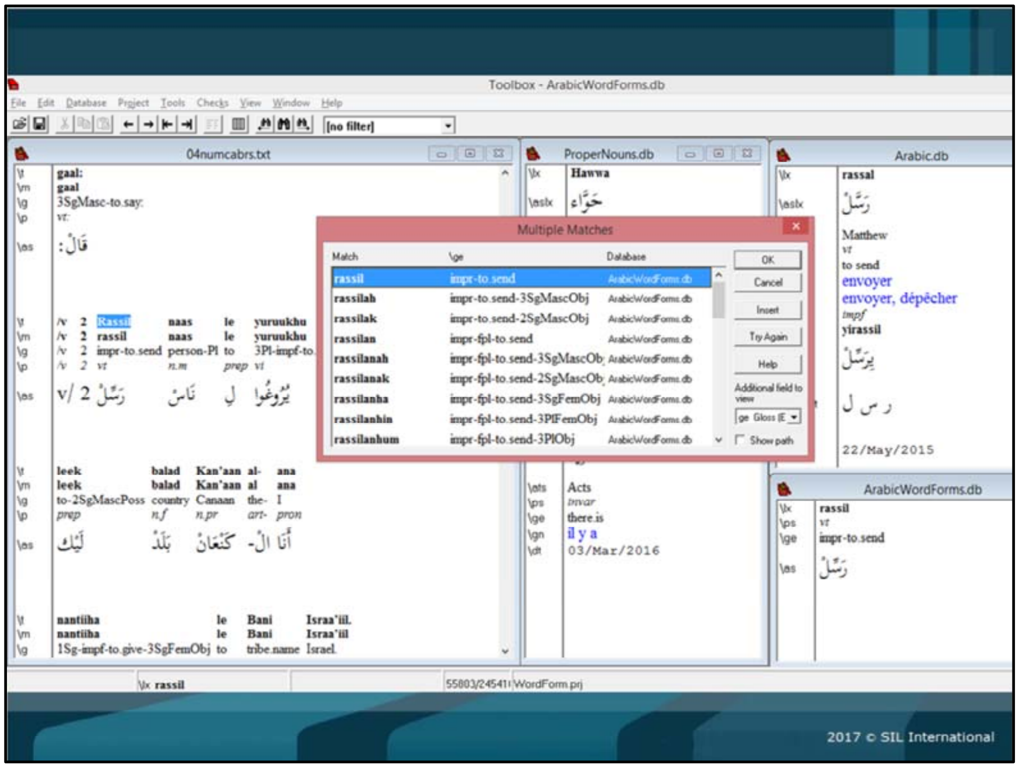
```
\lx    rassal
\ps    vt
\ge    3SgMasc-to.send
\as    رَسَّلْ
```

\ref 13   14/37   WordForm.prj

2017 © SIL International

Toolbox looks up all of those word forms in the sentence, and produces an interlinear that includes the Arabic script. And once the interlinear is complete, we can just extract the \as lines and we will have the conversion of the text in Arabic script.

I've highlighted the word "rassal" here, and you can see the source lexical entry to the right, as well as the specific word form that was used at the bottom. If we click the Interlinearize button a couple more times, we get to verse 2, which starts with the word "Rassil". That's the imperative form of this verb.



Toolbox - ArabicWordForms.db

04numcabrs.txt

```
\t    gaal:
\m    gaal
\g    3SgMasc-to.say:
\p    vt:
\as   قَالْ:

\t    /v 2 Rassil       naas       le    yuruukhu
\m    /v 2 rassil       naas       le    yuruukhu
\g    /v 2 impr-to.send person-Pl  to    3Pl-impf-to.
\p    /v 2 vt           n.m        prep  vi
\as   v/ 2 رَسِّلْ         نَاسْ        لِ     يْرُوغُوا

\t    leek             balad    Kan'aan  al-   ana
\m    leek             balad    Kan'aan  al    ana
\g    to-2SgMascPoss   country  Canaan   the-  I
\p    prep             n.f      n.pr     art-  pron
\as   لَيْك               بَلَدْ      كَنْعَانْ    الْ-   أَنَا

\t    nantiiha                 le   Bani        Israa'iil.
\m    nantiiha                 le   Bani        Israa'iil
\g    1Sg-impf-to.give-3SgFemObj to  tribe.name  Israel.
```

ProperNouns.db

```
\lx    Hawwa
\aslx  حَوَّاء

\ats   Acts
\ps    bivar
\ge    there.is
\gn    il y a
\dt    03/Mar/2016
```

Multiple Matches

Match	\ge	Database
rassil	impr-to.send	ArabicWordForms.db
rassilah	impr-to.send-3SgMascObj	ArabicWordForms.db
rassilak	impr-to.send-2SgMascObj	ArabicWordForms.db
rassilan	impr-fpl-to.send	ArabicWordForms.db
rassilanah	impr-fpl-to.send-3SgMascOb	ArabicWordForms.db
rassilanak	impr-fpl-to.send-2SgMascOb	ArabicWordForms.db
rassilanha	impr-fpl-to.send-3SgFemObj	ArabicWordForms.db
rassilanhin	impr-fpl-to.send-3PlFemObj	ArabicWordForms.db
rassilanhum	impr-fpl-to.send-3PlObj	ArabicWordForms.db

Arabic.db

```
\lx    rassal
\aslx  رَسَّلْ
       Matthew
       vt
       to send
       envoyer
       envoyer, dépêcher
       impf
       yirassil
       يرَسِّلْ
       ر س ل
       22/May/2015
```

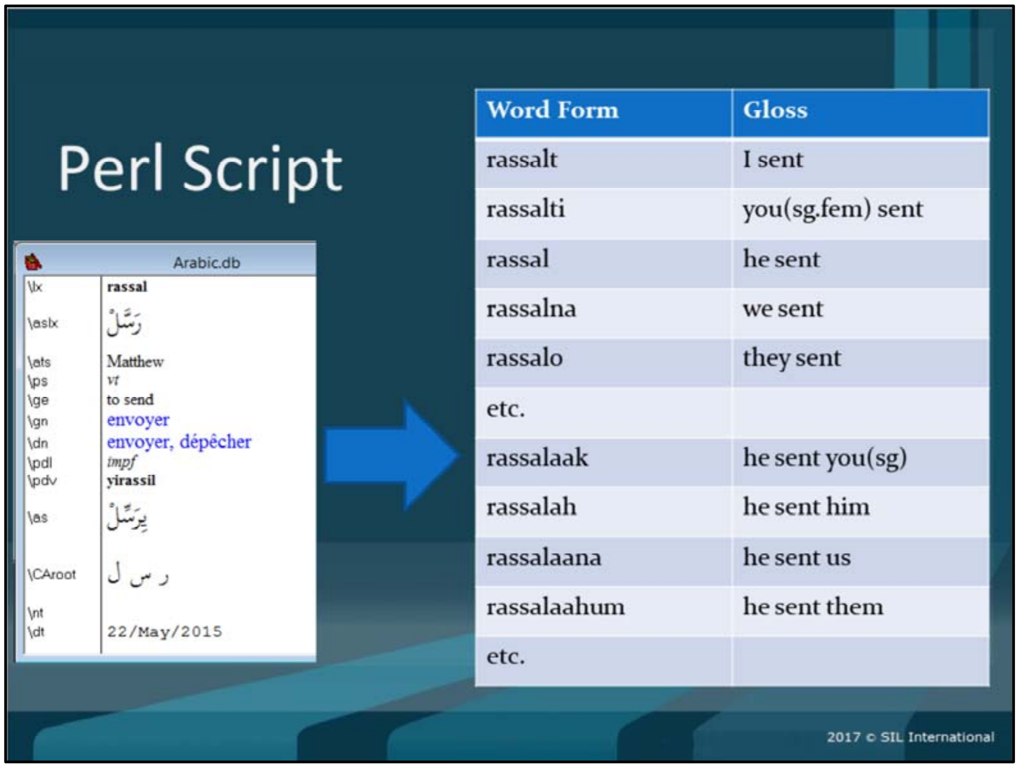
ArabicWordForms.db

```
\lx   rassil
\ps   vt
\ge   impr-to.send
\as   رَسِّلْ
```

\lx rassil   55803/245411 WordForm.prj

2017 © SIL International

What if this magical word forms database had that form of the word as well – Toolbox could just do the lookup in the same way to produce the interlinear. So now the question is how can we produce this magical word forms database, with all of the different possible word forms in Chadian Arabic? You probably can't read it at the bottom, but this word forms database has 245,000 word forms! When we search for "rassil", we not only get the imperative "Send!", but we also get forms with various object pronouns attached, like "Send him!" and "Send us!". (I might also mention that you also get some forms that don't make sense, like the command "Send you!", which is grammatical, but probably isn't ever used.) But how can we produce that database from our basic lexicon of less than 4,000 entries?



Perl Script

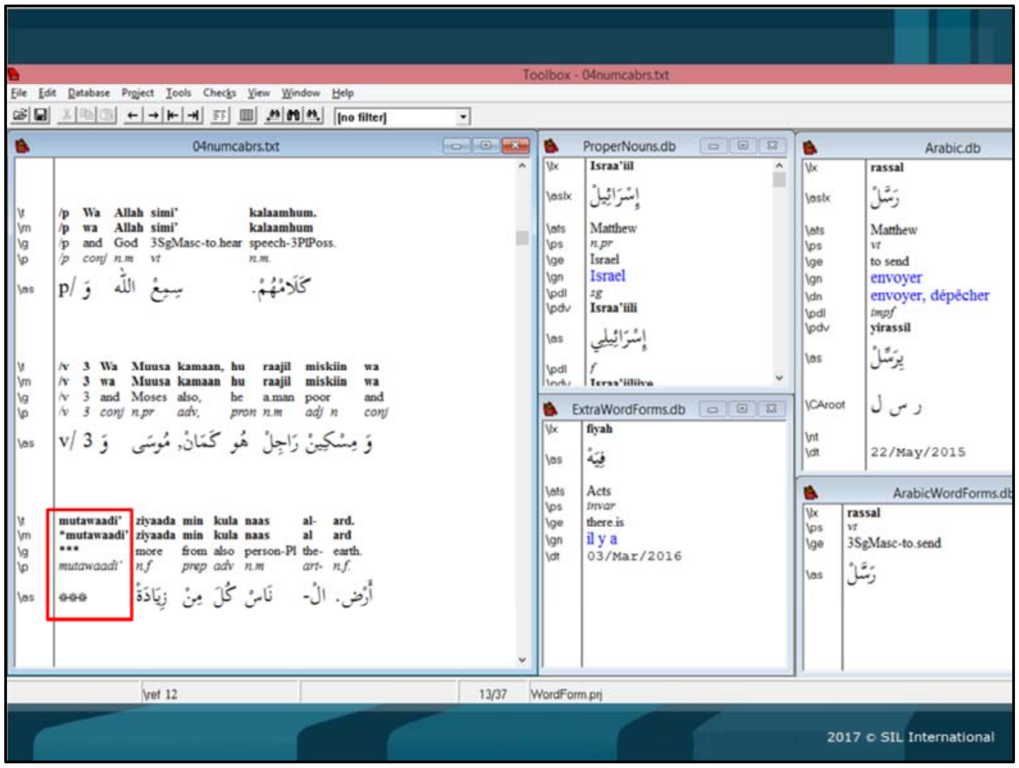
Arabic.db

\lx	rassal
\aslx	رَسَّلْ
\ats	Matthew
\ps	vt
\ge	to send
\gn	envoyer
\dn	envoyer, dépêcher
\pdl	impf
\pdv	yirassil
\as	يِرَسِّل
\CAroot	ر س ل
\nt	
\dt	22/May/2015

Word Form	Gloss
rassalt	I sent
rassalti	you(sg.fem) sent
rassal	he sent
rassalna	we sent
rassalo	they sent
etc.	
rassalaak	he sent you(sg)
rassalah	he sent him
rassalaana	he sent us
rassalaahum	he sent them
etc.	

2017 © SIL International

In this case the magic happens in a Perl script. Perl is a programming language similar to Python, with a lot of the same capabilities, like regular expressions. I believe the general opinion in SIL is that Python scripts are easier to read, but when I started this project, Python wasn't widely used yet, so I wrote it in Perl.
□ This Perl script takes each individual lexical entry, and, with an intimate knowledge of the grammar of Chadian Arabic, produces all of the possible word forms. "I sent", "he sent", "they sent"… and from each of those forms we add all of the possible object pronouns: "he sent you", "he sent us", "he sent them", etc. From a single entry in the lexicon, we can produce several hundred word forms.
And when I say the Perl script has an intimate knowledge of the grammar of Chadian Arabic, I mean intimate! When you attach affixes or object pronouns, you get all sorts of morphophonemic fireworks – vowel dropping and insertion, vocalic attraction, reordering, etc. And since the Perl script is building all of these word forms, it needs to know exactly what to do and when.



Toolbox - 04numcabrs.txt

04numcabrs.txt

```
\t   /p  Wa    Allah  simi'             kalaamhum.
\m   /p  wa    Allah  simi'             kalaamhum
\g   /p  and   God    3SgMasc-to.hear   speech-3PlPoss.
\p   /p  conj  n.m    vt                n.m.
\as  p/  وَ    اللّٰه  سِمِعْ              كَلَامْهُمْ.

\t   /v  3  Wa    Muusa  kamaan,  hu    raajil  miskiin  wa
\m   /v  3  wa    Muusa  kamaan   hu    raajil  miskiin  wa
\g   /v  3  and   Moses  also,    he    a.man   poor     and
\p   /v  3  conj  n.pr   adv,     pron  n.m     adj n    conj
\as  v/  3  وَ    مُوسَى  كَمَانْ,   هُو   رَاجِلْ   مِسْكِينْ  وَ

\t   mutawaadi'   ziyaada  min   kula  naas       al-   ard.
\m   *mutawaadi'  ziyaada  min   kula  naas       al    ard
\g   ***          more     from  also  person-Pl  the-  earth.
\p   mutawaadi'   n.f      prep  adv   n.m        art-  n.f.
\as  ~~***~~      زِيَادَةْ   مِنْ   كُلّ   نَاسْ       الْ-   أرْض.
```

ProperNouns.db

```
\lx    Israa'iil
\aslx  إِسْرَائِيل
\ats   Matthew
\ps    n.pr
\ge    Israel
\gn    Israel
\pdl   sg
\pdv   Israa'iili
\as    إِسْرَائِيلِي
\pdl   f
\pdv   Israa'iiliya
```

ExtraWordForms.db

```
\lx   fiyah
\as   فِيَهْ
\ats  Acts
\ps   invar
\ge   there.is
\gn   il y a
\dt   03/Mar/2016
```

Arabic.db

```
\lx      rassal
\aslx    رَسَّل
\ats     Matthew
\ps      vt
\ge      to send
\gn      envoyer
\dn      envoyer, dépêcher
\pdl     impf
\pdv     yirassil
\as      يِرَسِّل
\CAroot  ر س ل
\nt
\dt      22/May/2015
```

ArabicWordForms.db

```
\lx  rassal
\ps  vt
\ge  3SgMasc-to.send
\as  رَسَّل
```

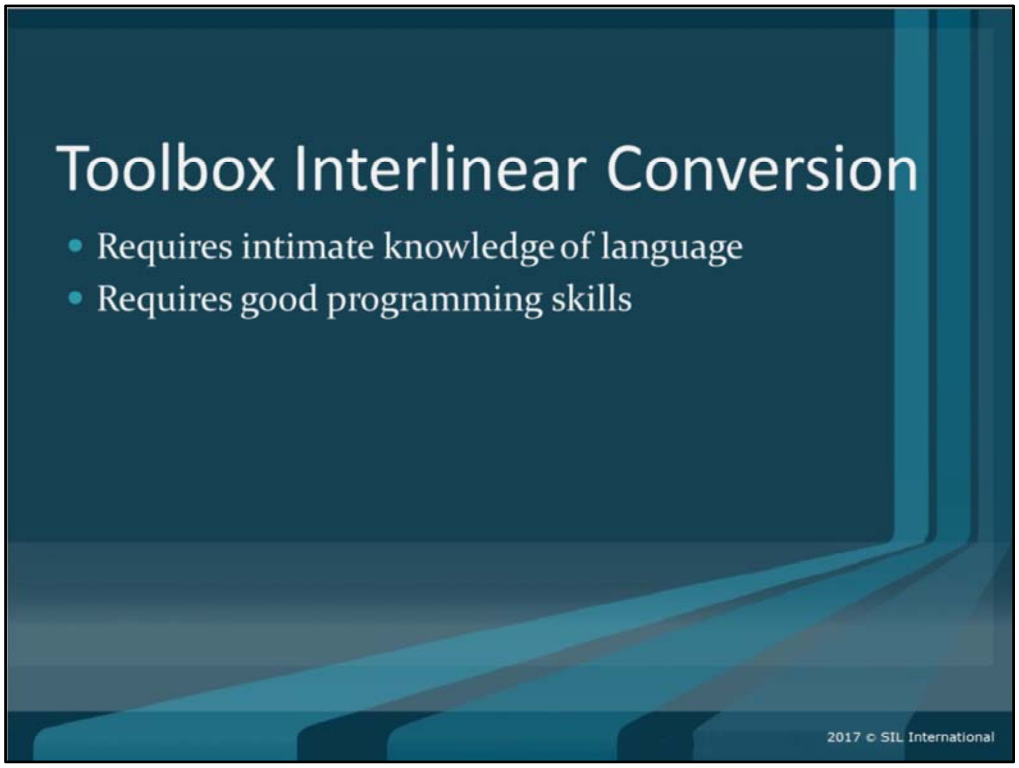
\ref 12 | 13/37 | WordForm.prj

2017 © SIL International

So the Perl script ends up producing this word forms database, which is basically a lookup table, with the RS and AS versions of each word form. And once you have all of those word forms, a simple interlinear configuration in Toolbox provides a quick and easy way to convert a Roman script text to Arabic script. What do you think happens if the text has a word that is not in the word forms database?

□ The interlinear lookup fails and usually puts out 3 asterisks. That usually that means that this word is missing in the lexicon, so the user can add it in, re-run the Perl script and try again. It could also mean that we don't have our grammar rules quite right, so we can try to tweak them. Or maybe this word is just an exception to the rules – language is messy, you know! In that case, we can add it to the Extra Word Forms database in the middle here, so that it can be matched manually.

□ How many of you have done linguistic analysis with something like Carla on an agglutinative language? Depending on the language, it can be fairly difficult to tease the morphemes apart. I was struggling with that in Chadian Arabic a number of years ago, and someone suggested that it might be easier to build words up from their component parts rather than trying to break word forms down into their component parts. And that gave birth to this project, where the Perl script produces all of the possible word forms from the lexical entries and lists of possible affixes, and those complete word forms are used for the analysis and conversion.

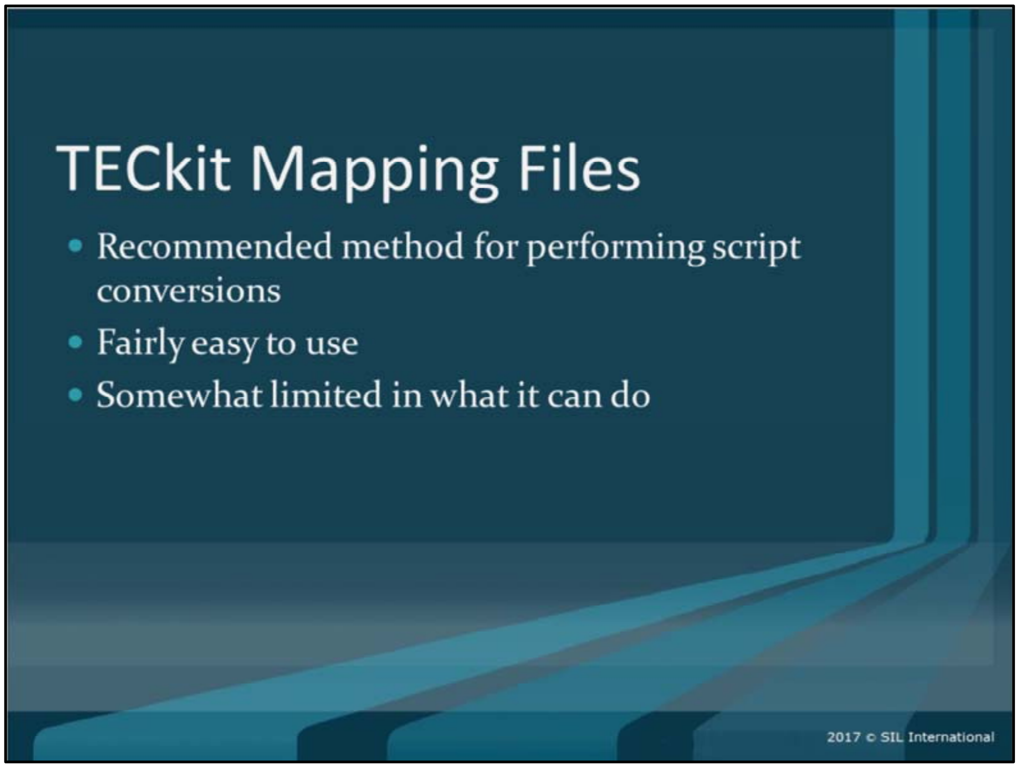


Toolbox Interlinear Conversion

- Requires intimate knowledge of language
- Requires good programming skills

2017 © SIL International

Just to make it clear… this conversion option is not for the faint of heart – it requires an intimate knowledge of the language and good programming skills. But in some cases, like Chadian Arabic, it might be the best option available.



TECkit Mapping Files

- Recommended method for performing script conversions
- Fairly easy to use
- Somewhat limited in what it can do

So even though I have shown you a couple of other options, most of you who are interested in script conversions will be creating TECkit mapping files for that purpose. And we're looking forward to helping you create those mapping files this week.
Are there any other questions?