ucharclasses

Mike "Pomax" Kamermans

December 19, 2010

Contents

1 introduction **2**

2 Use **2**
2.1 overriding ucharclass transitions 3

3 Load options **3**

4 Commands **4**
4.1 \setTransitionTo[2] 4
4.2 \setTransitionFrom[2] 4
4.3 \setTransitions[3] 4
4.4 \setTransitionsForXXXX[2] 4
4.5 \setDefaultTransitions[2] 4

5 Code **5**

6 Unicode blocks **7**

1 introduction

Sometimes you don't want to have to bother with font switching just because you're using languages that are distinct enough to use different unicode blocks, but aren't covered by the polyglossia package. Where normal word processing packages such as Microsoft Word or OpenOffice Writer handle this for you, LaTeX (because it needs you to tell it what to do) has no default behaviour for this, and so we arrive at a need for a package that does this for us. You probably already discovered that regular LaTeX has no understanding of unicode, and ended up going for Xe(La)TeX as your TeX compiler of choice, which means you now have two excellent resources available: fontspec, and ucharclasses.

The first of these lets you pick fonts based on what your system calls them, without needing to do anything special such as turning them into metafonts. This is convenient, user friendly, and the way of the future (at least, it should be). The second lets you define what should happen when your text changes from a character in one unicode block, to a character in another. This is also convenient, and paired with fontspec it offers automatic fontswitching in the same way that normal Office applications perform, with one big difference: you stay in control. If at some point you need the switch rule to do something different, you will be out of luck using an Office application. In Xe(La)TeX, you stay on top of things and still get to say exactly what happens, and when.

For instance, this document has no explicit font codes in the text itself, anywhere. Instead there are a few unicode block transition rules defined, which all say "when entering block ..., use fontspec to change the font to ...". As such, the following gibberish texts all just work:

- English: hfowehfoierfhioe (using Palatino Linotype)
- Arabic: كطبعز١٢٣٤٥ (using Tahoma and RTL/LTR rules)
- Japanese: 飽コネ路は向声ふ語言 (using Ume Mincho)
- Chinese: 么詳盉埳庂凡舟匌 (Ume Mincho when overlapping Japanese, SimSun for the rest)
- Hebrew: קדהפסבכֿךּלֿדָעֽ (using Times New Roman)
- Thai: ฦษฤยคซหถฆบชตท (using IrisUPC)
- Canadian first nations: ᐧᑕᓇᒍᙠᙣᙡᕗᐨ (using Code2000)
- Sinhala: ඞඥඊෆඍජළඎඩශ෴ (using Iskoola Pota)
- Malayalam: ഡൌഘലഴഒംള്ളഞൗമധ൦ (using Arial Unicode MS)
- and even domino tiles, 🀹🁁🁉🁑🁙🁛 (using Code2001), and mahjong tiles: 🀐🀑🀕🀗🀅 (using Symbola)

However, be aware that this automatic switching only works across unicode blocks. If you are working with typographically overlapping languages, such as combining English and Vietnamese in one document, things get a lot more complex if you want one font for English text and another for Vietnamese text. Both of these languagese use Latin blocks, so it is inherently impossible to tell which "language" a letter belongs to.

2 Use

In order to generate this document, the following preamble was used:

```
\usepackage{ucharclasses}
\usepackage{fontspec}
\usepackage{bidi}

\setDefaultTransitions{\fontspec{Code2000}}{}
\setTransitionsForLatin{\fontspec{Palatino Linotype}}{}
\setTransitionsForArabic{\fontspec{Tahoma}\setRTL}{\setLTR}
\setTransitionsForCJK{\fontspec{SimSun}{}
\setTransitionsForJapanese{\fontspec{Ume Mincho}}{}

\setTransitionTo{CJKUnifiedIdeographsExtensionB}{\fontspec{SimSun-ExtB}}
\setTransitionTo{Hebrew}{\fontspec{Times New Roman}}
\setTransitionTo{Thai}{\fontspec{IrisUPC}}
\setTransitionTo{Sinhala}{\fontspec{Iskoola Pota}}
\setTransitionTo{Malayalam}{\fontspec{Arial Unicode MS}}
```

```
\setTransitionTo{DominoTiles}{\fontspec{Code2001}}
\setTransitionTo{MahjongTiles}{\fontspec{Symbola}}
```

By default, ucharclasses is agnostic with regards to what you want to at the start or end of unicode blocks, so while using it for fontswitching is the most obvious application, you could also use it for far more creative purposes. An example of this is the Arabic block, which doesn't just need a different font, but also needs the text direction reversed, as it is read right-to-left, rather than left-to-right.

2.1 overriding ucharclass transitions

If you need to "override" ucharclass transition rules (for instance, you want a custom font for a bit of cross-unicode-block text), you will want to temporarily disable and reenabled XeTeX's interchartoks state. You can do this in three ways:

1. call [\XeTeXinterchartokstate = 0] before, and [\XeTeXinterchartokstate = 1] after you're done,
2. call the macros \disableTransitionRules before, and \enableTransitionRules after you're done, or
3. call \uccoff before, and \uccon after you're done.

This last option is mainly there because it's nice and short, and is more convenient in a scoped environment {\uccoff such as this\uccon} where you only want to override the transition behaviour within a paragraph. If you need it disabled for a few blocks of text instead, the full name commands are probably a better choice, because it makes your .tex more readable. As the base XeTeX command uses the unLaTeXy "... = ..." construction, it's best to avoid it outside of the preamble (and when using ucharclasses, should not be in the preamble at all).

3 Load options

Without any load options the ucharclasses package will simply set up transition rules for every Unicode block it knows of (currently that's Unicode 6.0). However, you can override this behaviour by explicitly indicating only those unicode blocks that you need for your document. For instance, the following will set up only block awareness for the Greek Unicode blocks:

\usepackage[GreekAndCoptic,GreekExtended]{ucharclasses}

In addition to allowing specific Unicode blocks, informal groups can also be used, so that the following will turn on awareness of all the CJK Unicode blocks:

\usepackage[CJK]{ucharclasses}

Currently available informal blocks are:

- Arabic
- Chinese
- CJK
- Cyrillic
- Diacritics
- Greek
- Korean
- Japanese
- Latin
- Mathematics
- Phonetics
- Punctuation
- Symbols
- Yi
- Other

4 Commands

4.1 \setTransitionTo[2]

This command has two arguments:

1. The name of the unicode class to which the transition should apply (see 'Unicode blocks' list)
2. The code you want used when entering this unicode block

4.2 \setTransitionFrom[2]

This command has two arguments:

1. The name of the unicode class to which the transition should apply (see 'Unicode blocks' list)
2. The code you want used when exiting this unicode block

4.3 \setTransitions[3]

This command has three arguments:

1. The name of the unicode class to which the transition should apply (see 'Unicode blocks' list)
2. The code you want used when entering this unicode block
3. The code you want used when exiting this unicode block

4.4 \setTransitionsForXXXX[2]

There are a number of these commands, pertaining to particular "informal groups": collections of unicode blocks which can be considered part of a single meta-block. Currently available informal groups (the names of which replace the XXXX in the section-stated command) are:

- Arabic
- Chinese
- CJK
- Cyrillic
- Diacritics
- Greek
- Korean
- Japanese
- Latin
- Mathematics
- Phonetics
- Punctuation
- Symbols
- Yi
- Other

Furthermore, these commands have two arguments:

1. The code you want used when entering blocks from the command's informal group
2. The code you want used when exiting blocks from the command's informal group

4.5 \setDefaultTransitions[2]

This is a blanket command that lets you set up the same to and from transition rules for all blocks in one go. It has (fairly obviously) two arguments:

1. The code you want used when entering any unicode block
2. The code you want used when exiting any unicode block

5 Code

The code relies on individual tex files that define the unicode blocks, each starting with:

```
\newcommand{\UnicodeBlockNameClass}{somenumber}
```

The classes are automatically numbered, per run, by using the \newXeTeXintercharclass command. The code then simply iterates over the class numbers from lower bound to upper bound; lower bound is recorded by calling \newXeTeXintercharclass once, before loading all the unicode blocks. Every block that is loaded then increments the class counter by calling \newXeTeXintercharclass for its class, and after the loading is done \newXeTeXintercharclass is called once more to get the iteration upper bound. This technically wastes two class numbers, but this is fairly irrelevant given the number of classes we use for all the different unicode blocks.

The block loading code is defined as follows:

```
\newcounter{glyphcounter}
\newcommand{\@defineUnicodeClass}[3]{
  \newXeTeXintercharclass#1
  \forloop{glyphcounter}{#2}{\value{glyphcounter}<#3}{
    \XeTeXcharclass\value{glyphcounter}=#1}
  \XeTeXcharclass#3=#1}

\newXeTeXintercharclass\@classstart
\@defineUnicodeClass{\AegeanNumbersClass}{65792}{65855}
...
\@defineUnicodeClass{\YijingHexagramSymbolsClass}{19904}{19967}
\newXeTeXintercharclass\@classend
```

And the transition commands are defined as follows:

```
\newcommand{\setTransitionsFor}[3]{
  \forloop{iclass}{he\@classstart}{\value{iclass} < \@nameuse{#1Class}}{
    \@transition{he\value{iclass}}{\@nameuse{#1Class}}{{#2}}
    \@transition{\@nameuse{#1Class}}{he\value{iclass}}{{#3}}}
  \addtocounter{iclass}{2}
  \forloop{iclass}{\value{iclass}}{\value{iclass} < he\@classend}{
    \@transition{he\value{iclass}}{\@nameuse{#1Class}}{{#2}}
    \@transition{\@nameuse{#1Class}}{he\value{iclass}}{{#3}}}
  % and a binding for the transitions to and from boundary characters
  \@transition{255}{\@nameuse{#1Class}}{{#2}}
  \@transition{\@nameuse{#1Class}}{255}{{#3}}}

\newcommand{\setTransitionTo}[2] {
  \forloop{iclass}{\the\@classstart}{\value{iclass} < \@nameuse{#1Class}}{
    \@transition{\the\value{iclass}}{\@nameuse{#1Class}}{{#2}}}
  \addtocounter{iclass}{2}
  \forloop{iclass}{\value{iclass}}{\value{iclass} < \the\@classend}{
    \@transition{\the\value{iclass}}{\@nameuse{#1Class}}{{#2}}}
  % and a binding for the transition from boundary characters
  \@transition{255}{\@nameuse{#1Class}}{{#2}}}

\newcommand{\setTransitionFrom}[2]{
  \forloop{iclass}{\the\@classstart}{\value{iclass} < \@nameuse{#1Class}}{
    \@transition{\@nameuse{#1Class}}{\the\value{iclass}}{{#2}}}
  \addtocounter{iclass}{2}
  \forloop{iclass}{\value{iclass}}{\value{iclass} < \the\@classend}{
    \@transition{\@nameuse{#1Class}}{\the\value{iclass}}{{#2}}}
  % and a binding for the transition to boundary characters
  \@transition{\@nameuse{#1Class}}{255}{{#2}}}
```

The broad level \setTransitionsFor(InformalGroupName)[2] commands are essentially wrapper commands, calling \setTransitionsFor for each blocks that is in the informal group. For Arabic, for instance, the code is:

```
\newcommand{\setTransitionsForArabic}[2]{
  \setTransitionsFor{Arabic}{#1}{#2}
  \setTransitionsFor{ArabicPresentationFormsA}{#1}{#2}
  \setTransitionsFor{ArabicPresentationFormsB}{#1}{#2}
  \setTransitionsFor{ArabicSupplement}{#1}{#2}
}
```

6 Unicode blocks

The following Unicode blocks are currently available:

- AegeanNumbers
- AlphabeticPresentationForms
- AncientGreekMusicalNotation
- AncientGreekNumbers
- AncientSymbols
- Arabic
- ArabicPresentationFormsA
- ArabicPresentationFormsB
- ArabicSupplement
- Armenian
- Arrows
- Balinese
- BasicLatin
- Bengali
- BlockElements
- Bopomofo
- BopomofoExtended
- BoxDrawing
- BraillePatterns
- Buginese
- Buhid
- ByzantineMusicalSymbols
- Carian
- Cham
- Cherokee
- CJKCompatibility
- CJKCompatibilityForms
- CJKCompatibilityIdeographs
- CJKCompatibilityIdeographsSupplement
- CJKRadicalsSupplement
- CJKStrokes
- CJKSymbolsandPunctuation
- CJKUnifiedIdeographs
- CJKUnifiedIdeographsExtensionA
- CJKUnifiedIdeographsExtensionB
- CombiningDiacriticalMarks
- CombiningDiacriticalMarksforSymbols
- CombiningDiacriticalMarksSupplement
- CombiningHalfMarks
- ControlPictures
- Coptic
- CountingRodNumerals
- Cuneiform
- CuneiformNumbersandPunctuation
- CurrencySymbols
- CypriotSyllabary
- Cyrillic
- CyrillicExtendedA
- CyrillicExtendedB
- CyrillicSupplement
- Deseret
- Devanagari
- Dingbats
- DominoTiles
- EnclosedAlphanumerics
- EnclosedCJKLettersandMonths
- Ethiopic
- EthiopicExtended
- EthiopicSupplement
- GeneralPunctuation
- GeometricShapes
- Georgian
- GeorgianSupplement
- Glagolitic
- Gothic
- GreekandCoptic
- GreekExtended
- Gujarati
- Gurmukhi
- HalfwidthandFullwidthForms
- HangulCompatibilityJamo
- HangulJamo
- HangulSyllables
- Hanunoo
- Hebrew
- Hiragana
- IdeographicDescriptionCharacters
- IPAExtensions
- Kanbun
- KangxiRadicals
- Kannada
- Katakana
- KatakanaPhoneticExtensions
- KayahLi
- Kharoshthi
- Khmer
- KhmerSymbols
- Lao
- LatinExtendedA
- LatinExtendedAdditional
- LatinExtendedB
- LatinExtendedC
- LatinExtendedD
- LatinSupplement
- Lepcha
- LetterlikeSymbols
- Limbu
- LinearBIdeograms
- LinearBSyllabary
- loadblocks.tex
- Lycian
- Lydian
- MahjongTiles
- Malayalam
- MathematicalAlphanumericSymbols
- MathematicalOperators
- MiscellaneousMathematicalSymbolsA
- MiscellaneousMathematicalSymbolsB

- MiscellaneousSymbolsandArrows
- MiscellaneousSymbols
- MiscellaneousTechnical
- ModifierToneLetters
- Mongolian
- moo.txt
- MusicalSymbols
- Myanmar
- NewTaiLue
- NKo
- NumberForms
- Ogham
- OldChiki
- OldItalic
- OldPersian
- OpticalCharacterRecognition
- Oriya
- Osmanya
- PhagsPa
- PhaistosDisc
- Phoenician
- PhoneticExtensions
- PhoneticExtensionsSupplement
- PrivateUseArea
- Rejang
- Runic
- Saurashtra
- Shavian
- Sinhala
- SmallFormVariants
- SpacingModifierLetters
- Specials
- SuperscriptsandSubscripts
- SupplementalArrowsA
- SupplementalArrowsB
- SupplementalMathematicalOperators
- SupplementalPunctuation
- SupplementaryPrivateUseAreaA
- SupplementaryPrivateUseAreaB
- SylotiNagri
- Syriac
- Tagalog
- Tagbanwa
- Tags
- TaiLe
- TaiXuanJingSymbols
- Tamil
- Telugu
- Thaana
- Thai
- Tibetan
- Tifinagh
- Ugaritic
- UnifiedCanadianAboriginalSyllabics
- Vai
- VariationSelectors
- VariationSelectorsSupplement
- VerticalForms
- YijingHexagramSymbols
- YiRadicals
- YiSyllables