Documentation for package `interchar` *

Zou Hu (zohooo@yeah.net)

February 17, 2015

Contents

1 Introduction

With XeTeX's character class mechanism, we could put characters into different classes, and insert some tokens to the input stream between characters in one class and those in another class. This mechanism is originally used for switching fonts and adjusting spaces. But it has many other useful applications.

By default, all characters are of class 0, except CJK ideographs are of class 1, CJK left punctuation of class 2, CJK right punctuation of class 3, text boundaries (glues, kerns, maths, boxes, etc.) of class 255, and several characters of class 256 (this class is ignored).

Package interchar is written for making character class mechanism more easy to use. For example, after loading the package with `\usepackage{interchar}`, you may change the color of every occurrences of character o:

```
\newintercharclass{\myclass}
\intercharclass{`\o}{\myclass}
\interchartoks{0}{\myclass}{\bgroup\color{red}}
\interchartoks{255}{\myclass}{\bgroup\color{red}}
\interchartoks{\myclass}{0}{\egroup}
\interchartoks{\myclass}{255}{\egroup}
\intercharstate{1}
```

There are some existing packages using this mechanism, such as polyglossia, xeCJK and xesearch. But since one character could only be put into one single class, when loading two or more of these packages simultaneously, it would be very likely that some conflicts occur.

Package interchar also provides some migration commands for these packages. With minor changes these packages should be compatible with interchar, and users could switch between different character class schemes when loading these packages at the same time.

*Version 0.2. Please report bugs via https://github.com/zohooo/interchar.

2 Commands for normal users

- `\newintercharscheme{`⟨*scheme*⟩`}` creates a new char class scheme. For example,

  ```
  \newintercharscheme{foo}
  ```

  There is a prebuilt `default` scheme which you need not to create it. The first argument of the other commands in this section is optional; its default value is `default`.

- `\intercharstate[`⟨*scheme*⟩`]{`⟨*state-code*⟩`}` changes current scheme. If ⟨*state-code*⟩ is positive, changes current scheme to ⟨*scheme*⟩; otherwise, changes current scheme to `default`. For example,

  ```
  \intercharstate[foo]{1}
  ```

- `\getintercharstate[`⟨*scheme*⟩`]{\cs}` gets current state of the specified scheme and store the state code in `\cs`. For example,

  ```
  \getintercharstate[foo]{\mystate}
  ```

- `\newintercharclass[`⟨*scheme*⟩`]{\cs}` creates a new char class in the specified scheme and stores the class number in `\cs`. For example,

  ```
  \newintercharclass[foo]{\myclass}
  ```

- `\intercharclass[`⟨*scheme*⟩`]{`⟨*char-range*⟩`}{`⟨*char-class*⟩`}` moves all characters within ⟨*char-range*⟩ to the class ⟨*char-class*⟩ in the specified scheme. For example,

  ```
  \intercharclass[foo]{`\@}{255}
  \intercharclass[foo]{`\a-`\z}{\myclass}
  ```

- `\getintercharclass[`⟨*scheme*⟩`]{`⟨*char-code*⟩`}{\cs}` gets the class number of the specified character in the specified scheme, and stores the result in `\cs`. For example,

  ```
  \getintercharclass[foo]{`\@}{\result}
  ```

- `\interchartoks[`⟨*scheme*⟩`]{`⟨*char-class-1*⟩`}{`⟨*char-class-2*⟩`}{`⟨*tokens*⟩`}` defines tokens to be inserted between ⟨*char-class-1*⟩ and ⟨*char-class-2*⟩ (in that order) in the specified scheme. For example,

  ```
  \interchartoks[foo]{0}{\myclass}{\bgroup\color{red}}
  \interchartoks[foo]{\myclass}{0}{\egroup}
  ```

- `\getinterchartoks[`⟨*scheme*⟩`]{`⟨*char-class-1*⟩`}{`⟨*char-class-2*⟩`}{\cs}` gets the tokens to be inserted between ⟨*char-class-1*⟩ and ⟨*char-class-2*⟩ in the specified scheme, and stores the result in `\cs`. For example,

  ```
  \getinterchartoks[foo]{0}{\myclass}{\mytoks}
  ```

3 Commands for macro writers

For macro writers, we provide `\NewIntercharScheme` command which is an alias of user command `\newintercharscheme`. When you write `\NewIntercharScheme{foo}`, interchar package also creates the following migration commands for you:

Migration Command	Analogous to Commands
`\FooIntercharState`␣=␣⟨*state-code*⟩	`\XeTeXinterchartokenstate`
`\GetFooIntercharState`	`\the\XeTeXinterchartokenstate`
`\NewFooIntercharClass`␣⟨*control-sequence*⟩	`\newXeTeXintercharclass`
`\FooIntercharClass`␣⟨*char-code*⟩␣=␣⟨*char-class*⟩	`\XeTeXcharclass`
`\GetFooIntercharClass`␣⟨*char-code*⟩	`\the\XeTeXcharclass`.
`\FooIntercharToks`␣⟨*class1*⟩␣⟨*class2*⟩␣=␣{⟨*toks*⟩}	`\XeTeXinterchartoks`
`\GetFooIntercharToks`␣⟨*class1*⟩␣⟨*class2*⟩	`\the\XeTeXinterchartoks`

Within the command specifications in the above table, all of the equal signs = and most of the spaces are optional.

If you are the author of package foo which use XeTeX's char class mechanism, with some minor changes you could make your package compatible with other packages.

1. Add the following lines to the beginning of your package file:

```
\ifdefined\NewIntercharScheme
    \NewIntercharScheme{foo}%
\else
    \let \FooIntercharState = \XeTeXinterchartokenstate
    \def \GetFooIntercharState {\the\XeTeXinterchartokenstate}%
    \let \NewFooIntercharClass = \newXeTeXintercharclass
    \let \FooIntercharClass = \XeTeXcharclass
    \def \GetFooIntercharClass {\the\XeTeXcharclass}%
    \let \FooIntercharToks = \XeTeXinterchartoks
    \def \GetFooIntercharToks {\the\XeTeXinterchartoks}%
\fi
```

2. Rename every occurrence of XeTeX's commands in your package file with the new name according to the above table.

After these modifications, when users doesn't load interchar, your package should behave as before, but when users load interchar package **before** your package, interchar will takes over XeTeX's char class mechanism, therefore users could switch among different char class schemes at will.

Note that `\GetFooIntercharState` is **fully expandable**, which means you could write the following conditional test:

```
\ifnum \GetFooIntercharState > 0 doA \else doB \fi
```

However `\GetFooIntercharClass` and `\GetFooIntercharToks` are **not fully expandable** for the time being.

4 Implementation

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{interchar}[2015/02/16 v0.2
    managing character class schemes of XeTeX]
\RequirePackage{expl3}[2014/05/20]
\RequirePackage{xparse}
\ExplSyntaxOn
```

Some more scratch variables.

```
\tl_new:N \l__interchar_a_tl
\tl_new:N \l__interchar_b_tl
\tl_new:N \l__interchar_x_tl
\tl_new:N \l__interchar_y_tl
\tl_new:N \l__interchar_ab_tl
\tl_new:N \l__interchar_xy_tl
```

Generate variants for some functions.

```
\cs_generate_variant:Nn \clist_concat:NNN { c }
\cs_generate_variant:Nn \int_to_hexadecimal:n { V }
\cs_generate_variant:Nn \prop_get:NnN { cx }
\cs_generate_variant:Nn \prop_item:cn { cx }
\cs_generate_variant:Nn \prop_put:Nnn { cx }
\cs_generate_variant:Nn \tl_if_eq:nnT { Vo }
```

Rename some primitive commands of XeTeX.

```
\cs_new_eq:NN \xetex_intercharstate:D \XeTeXinterchartokenstate
\cs_new_eq:NN \xetex_newcharclass:D \newXeTeXintercharclass
\cs_new_eq:NN \xetex_charclass:D \XeTeXcharclass
\cs_new_eq:NN \xetex_interchartoks:D \XeTeXinterchartoks
```

Need to update them according to `unicode-letters.tex`.

```
\clist_new:N \g_interchar_default_classes_clist
\clist_gset:Nn \g_interchar_default_classes_clist { 1, 2, 3, 256 }
\int_new:N  \g_interchar_default_newclass_int
\int_gset:Nn \g_interchar_default_newclass_int { 4 }
\clist_new:c { l_interchar_default_chars_0_clist }
\clist_set:cn { l_interchar_default_chars_0_clist }
  {
    1-2319, 231C-2328, 232B-23EF, 23F4-25FF, 2604-2613, 2616-2617, 2619,
    2620-2638, 263C-2667, 2669-267E, 2680-26BC, 26C9-26CC, 26CE, 26D2,
    26D5-26D7, 26DA-26DB, 26DD-26DE, 26E2-26E9, 26EB-26F0, 26F6, 26FB-26FC,
    2705-2707, 270E-2E7F, 2E9A, 2EF4-2EFF, 2FD6-2FEF, 2FFC-3000, 3040-3041,
    3043, 3045, 3047, 3049, 3063, 3083, 3085, 3087, 308E, 3095-3098, 30A1, 30A3,
    30A5, 30A7, 30A9, 30C3, 30E3, 30E5, 30E7, 30EE, 30F5-30F6, 30FC, 3100-3104,
    312E-3130, 318F, 31BB-31BF, 31E4-31FF, 321F, 3248-324F, 32FF, 4DC0-4DFF,
    A48D-A48F, A4C7-F8FF, FB00-FE0F, FE19-FE2F, FE53, FE67, FE69-FE6A,
    FE6C-FF00, FF04-FF05, FF66-FF9D, FFA0-FFE1, FFE5-FFFD
  }
\clist_new:c { l_interchar_default_chars_1_clist }
\clist_set:cn { l_interchar_default_chars_1_clist }
  {
    231A-231B, 23F0-23F3, 2600-2603, 2614-2615, 2618, 261A-261F, 2639-263B,
    2668, 267F, 26BD-26C8, 26CD, 26CF-26D1, 26D3-26D4, 26D8-26D9, 26DC,
```

```
      26DF-26E1, 26EA, 26F1-26F5, 26F7-26FA, 26FD-2704, 2708-270D, 2E80-2E99,
      2E9B-2EF3, 2F00-2FD5, 2FF0-2FFB, 3003-3004, 3006-3007, 3012-3013, 3020-3029,
      3030-3034, 3036-303A, 303D-303F, 3042, 3044, 3046, 3048, 304A-3062,
      3064-3082, 3084, 3086, 3088-308D, 308F-3094, 309F, 30A2, 30A4, 30A6, 30A8,
      30AA-30C2, 30C4-30E2, 30E4, 30E6, 30E8-30ED, 30EF-30F4, 30F7-30FA, 30FF,
      3105-312D, 3131-318E, 3190-31BA, 31C0-31E3, 3200-321E, 3220-3247, 3250-32FE,
      3300-4DBF, 4E00-A014, A016-A48C, A490-A4C6, F900-FAFF, FE30-FE34, FE45-FE46,
      FE49-FE4F, FE51, FE58, FE5F-FE66, FE68, FE6B, FF02-FF03, FF06-FF07,
      FF0A-FF0B, FF0D, FF0F-FF19, FF1C-FF1E, FF20-FF3A, FF3C, FF3E-FF5A, FF5C,
      FF5E, FFE2-FFE4, 1B000-1B001, 1F000-1F02B, 1F030-1F093, 1F0A0-1F0AE,
      1F0B1-1F0BF, 1F0C1-1F0CF, 1F0D1-1F0F5, 1F200-1F202, 1F210-1F23A,
      1F240-1F248, 1F250-1F251, 1F300-1F32C, 1F330-1F37D, 1F380-1F39B,
      1F39E-1F3B4, 1F3B7-1F3BB, 1F3BD-1F3CE, 1F3D4-1F3F7, 1F400-1F49F, 1F4A1,
      1F4A3, 1F4A5-1F4AE, 1F4B0, 1F4B3-1F4FE, 1F507-1F516, 1F525-1F531, 1F54A,
      1F550-1F579, 1F57B-1F5A3, 1F5A5-1F5D3, 1F5DC-1F5F3, 1F5FA-1F642,
      1F645-1F64F, 1F680-1F6CF, 1F6E0-1F6EC, 1F6F0-1F6F3,
    }
  \clist_new:c  { l_interchar_default_chars_2_clist }
  \clist_set:cn { l_interchar_default_chars_2_clist }
    {
      2329, 3008, 300A, 300C, 300E, 3010, 3014, 3016, 3018, 301A, 301D, FE17,
      FE35, FE37, FE39, FE3B, FE3D, FE3F, FE41, FE43, FE47, FE59, FE5B, FE5D,
      FF08, FF3B, FF5B, FF5F, FF62
    }
  \clist_new:c { l_interchar_default_chars_3_clist }
  \clist_set:cn { l_interchar_default_chars_3_clist }
    {
      232A, 3001-3002, 3005, 3009, 300B, 300D, 300F, 3011, 3015, 3017, 3019,
      301B-301C, 301E-301F, 303B-303C, 309B-309E, 30A0, 30FB, 30FD-30FE, A015,
      FE10-FE16, FE18, FE36, FE38, FE3A, FE3C, FE3E, FE40, FE42, FE44, FE48, FE50,
      FE52, FE54-FE57, FE5A, FE5C, FE5E, FF01, FF09, FF0C, FF0E, FF1A-FF1B, FF1F,
      FF3D, FF5D, FF60-FF61, FF63-FF65, FF9E-FF9F
    }
  \clist_set:cn { l_interchar_default_chars_256_clist }
    {
      302A-302F, 3035, 3099-309A
    }
  \prop_new:N \l_interchar_default_toks_prop
  \prop_put:Nnn \l_interchar_default_toks_prop {0~1} {\xtxHanSpace}
  \prop_put:Nnn \l_interchar_default_toks_prop {0~2} {\xtxHanSpace}
  \prop_put:Nnn \l_interchar_default_toks_prop {0~3} {\nobreak\xtxHanSpace}
  \prop_put:Nnn \l_interchar_default_toks_prop {1~0} {\xtxHanSpace}
  \prop_put:Nnn \l_interchar_default_toks_prop {2~0} {\nobreak\xtxHanSpace}
  \prop_put:Nnn \l_interchar_default_toks_prop {3~0} {\xtxHanSpace}
  \prop_put:Nnn \l_interchar_default_toks_prop {1~1} {\xtxHanGlue}
  \prop_put:Nnn \l_interchar_default_toks_prop {1~2} {\xtxHanGlue}
  \prop_put:Nnn \l_interchar_default_toks_prop {1~3} {\nobreak\xtxHanGlue}
  \prop_put:Nnn \l_interchar_default_toks_prop {2~1} {\nobreak\xtxHanGlue}
  \prop_put:Nnn \l_interchar_default_toks_prop {2~2} {\nobreak\xtxHanGlue}
  \prop_put:Nnn \l_interchar_default_toks_prop {2~3} {\xtxHanGlue}
  \prop_put:Nnn \l_interchar_default_toks_prop {3~1} {\xtxHanGlue}
  \prop_put:Nnn \l_interchar_default_toks_prop {3~2} {\xtxHanGlue}
  \prop_put:Nnn \l_interchar_default_toks_prop {3~3} {\nobreak\xtxHanGlue}
```

Create a new interchar scheme for each package.

```
  \msg_new:nnn { interchar } { Empty-Argument }
```

```
    {
      The~argument~should~not~be~empty!
    }
  \tl_new:N \l_interchar_current_scheme_tl
  \tl_set:Nn \l_interchar_current_scheme_tl {default}
  \NewDocumentCommand \newintercharscheme { m }
    {
      \tl_if_empty:nT {#1} { \msg_critical:nn { interchar } { Empty-Argument } }
      \clist_new:c { g_interchar_#1_classes_clist }
      \clist_gset:cn { g_interchar_#1_classes_clist } { 1, 2, 3 }
      \int_new:c { g_interchar_#1_newclass_int }
      \int_gset:cn { g_interchar_#1_newclass_int } { 4 }
      \clist_new:c { l_interchar_#1_chars_1_clist }
      \clist_new:c { l_interchar_#1_chars_2_clist }
      \clist_new:c { l_interchar_#1_chars_3_clist }
      \prop_new:c { l_interchar_#1_toks_prop }
      % Used for migrating from XeTeX's primitive commands
      \interchar@migration { #1 }
    }
```

High level `\intercharstate` command. `#1`: scheme name; `#2`: state code.

```
  \NewDocumentCommand \intercharstate { O{default} m }
    {
      \interchar_state:nn {#1} {#2}
    }
  \cs_new_protected_nopar:Npn \interchar_state:nn #1#2
    {
      \__interchar_clear_toks:V \l_interchar_current_scheme_tl
      \clist_map_inline:Nn \g_interchar_default_classes_clist
          { \__interchar_apply_class:nn {default} {##1} }
      \__interchar_apply_class:nn {default} {0}
      \__interchar_apply_toks:n {default}
      \int_compare:nTF { #2 > 0 }
        {
          \clist_map_inline:cn { g_interchar_#1_classes_clist }
              { \__interchar_apply_class:nn {#1} {##1} }
          \__interchar_apply_toks:n {#1}
          % Use \tl_set:Nx rathar than \tl_set:Nn here
          \tl_set:Nx \l_interchar_current_scheme_tl {#1}
        }
        { \tl_set:Nn \l_interchar_current_scheme_tl {default} }
    }
  \cs_generate_variant:Nn \interchar_state:nn { VV }
```

`#1`: scheme name; `#2`: class number.

```
  \cs_new_protected_nopar:Npn \__interchar_apply_class:nn #1#2
    {
      \clist_map_inline:cn
        { l_interchar_#1_chars_ \int_to_arabic:n{#2} _clist }
        {
          \__interchar_class_split_range:nNN {##1} \l_tmpa_tl \l_tmpb_tl
          \int_set:Nn \l_tmpa_int { "\l_tmpa_tl }
          \int_set:Nn \l_tmpb_int { "\l_tmpb_tl }
          \int_while_do:nn { \l_tmpa_int <= \l_tmpb_int }
            {
```

```
              \xetex_charclass:D \l_tmpa_int = #2
              \int_incr:N \l_tmpa_int
            }
        }
    }
```

`#1`: scheme name.

```
  \cs_new_protected_nopar:Npn \__interchar_apply_toks:n #1
    {
      \prop_map_inline:cn { l_interchar_#1_toks_prop }
        { \xetex_interchartoks:D ##1 = {##2} }
    }
```

`#1`: scheme name.

```
  \cs_new_protected_nopar:Npn \__interchar_clear_toks:n #1
    {
      \prop_map_inline:cn { l_interchar_#1_toks_prop }
        { \xetex_interchartoks:D ##1 = {} }
    }
  \cs_generate_variant:Nn \__interchar_clear_toks:n { V }
```

High level `\getintercharstate` command. `#1`: scheme name; `#2` result control sequence.

```
  \DeclareDocumentCommand \getintercharstate { O{default} m }
    {
      \interchar_get_state:nN {#1} {#2}
    }
  \cs_new_protected_nopar:Npn \interchar_get_state:nN #1#2
    {
      \tl_set:Nx #2 { \interchar_get_state:n {#1} }
    }
```

`#1`: scheme name. This function is fully expandable.

```
  \cs_new_nopar:Npn \interchar_get_state:n #1
    {
      \str_if_eq:VnTF \l_interchar_current_scheme_tl { #1 } { 1 } { 0 }
    }
```

High level `\intercharnewclass` command. `#1`: scheme name; `#2`: control sequence for class number.

```
  \NewDocumentCommand \newintercharclass { O{default} m}
    {
      \interchar_newclass:nn { #1 } { #2 }
    }
  \cs_new_protected_nopar:Npn \interchar_newclass:nn #1#2
    {
      \int_set:Nn \l_tmpa_int { \int_use:N \use:c {g_interchar_#1_newclass_int} }
      \int_new:N #2
      \int_set_eq:NN #2 \l_tmpa_int
      \clist_put_right:co {g_interchar_#1_classes_clist} {\int_use:N \l_tmpa_int}
      \clist_new:c { l_interchar_#1_chars_ \int_use:N\l_tmpa_int _clist }
      \int_incr:c { g_interchar_#1_newclass_int }
    }
```

High level `\intercharclass` command. `#1`: scheme name; `#2`: char range; `#3`: class number.

```
\NewDocumentCommand \intercharclass { O{default} m m }
  {
    \interchar_class:nnn {#1} {#2} {#3}
  }
\bool_new:N \g__interchar_class_delete_char_bool
\cs_new_protected_nopar:Npn \interchar_class:nnn #1#2#3
  {
    \int_compare:nT { #3 > 0 }
      {
        \int_set:Nn \l_tmpa_int {#3}
        \clist_if_in:coF {g_interchar_#1_classes_clist}
          { \int_use:N \l_tmpa_int }
          {
            \clist_put_right:co { g_interchar_#1_classes_clist }
                { \int_use:N\l_tmpa_int }
            \clist_new:c
                { l_interchar_#1_chars_ \int_use:N\l_tmpa_int _clist }
          }
        \__interchar_class_insert_char:nnn {#1} {#3} {#2}
      }
    \bool_set_false:N \g__interchar_class_delete_char_bool
    \clist_map_inline:cn {g_interchar_#1_classes_clist}
      {
        \int_compare:nT { #3 != ##1 }
          {
            \bool_if:NTF \g__interchar_class_delete_char_bool
              { \clist_map_break: }
              { \__interchar_class_delete_char:nnn {#1} {##1} {#2} }
          }
      }
  }
\cs_generate_variant:Nn \interchar_class:nnn { VVV }
```

High level `\getintercharclass` command. `#1`: scheme name; `#2`: char code; `#3`: result control sequence.

```
\DeclareDocumentCommand \getintercharclass { O{default} m m }
  {
    \interchar_get_class:nnN {#1} {#2} {#3}
  }
\cs_new_protected_nopar:Npn \interchar_get_class:nnN #1#2#3
  {
    \tl_set:Nx #3 { \interchar_get_class:nn {#1} {#2} }
  }
```

`#1`: scheme name; `#2`: char code. This function is fully expandable.

```
\cs_new_nopar:Npn \interchar_get_class:nn #1#2
  {
    \__interchar_get_class_aux:vNnn { g_interchar_#1_classes_clist }
        \__interchar_class_find_char:nnn {#1} {#2}
  }
\cs_generate_variant:Nn \interchar_get_class:nn { VV }
\cs_new_nopar:Npn \__interchar_get_class_aux:nNnn #1#2#3#4
  {
    \__interchar_get_class_loop:Nnnw #2 {#3} {#4}
         #1 , \q_recursion_tail , \q_recursion_stop
```

```
    }
  \cs_generate_variant:Nn \__interchar_get_class_aux:nNnn { v }
  \cs_new_nopar:Npn \__interchar_get_class_loop:Nnnw #1#2#3#4 ,
    {
      \quark_if_recursion_tail_stop:n {#4}
      #1 {#2} {#4} {#3}
      \__interchar_get_class_loop:Nnnw #1 {#2} {#3}
    }
```

#1: scheme name; #2: class number; #3: char code.

```
  \cs_new_nopar:Npn \__interchar_class_find_char:nnn #1#2#3
    {
      \__interchar_class_find_char_aux:vnn {l_interchar_#1_chars_#2_clist} {#2} {#3}
    }
  \cs_new_nopar:Npn \__interchar_class_find_char_aux:nnn #1#2#3
    {
      \__interchar_class_find_char_loop:nnw {#2} {#3}
           #1 , \q_recursion_tail , \q_recursion_stop
    }
  \cs_generate_variant:Nn \__interchar_class_find_char_aux:nnn { v }
  \cs_new_nopar:Npn \__interchar_class_find_char_loop:nnw #1#2#3 ,
    {
      \quark_if_recursion_tail_stop:n {#3}
      \int_case:nn { \__interchar_class_compare_char:nn {#2} {#3} }
        {
          { -1 } { \use_none_delimit_by_q_recursion_stop:w }
          { 0 } { #1 % found the char in class #1, stop two level loops
            \use_i_delimit_by_q_recursion_stop:nw
                { \use_none_delimit_by_q_recursion_stop:w }
          }
        }
      \__interchar_class_find_char_loop:nnw {#1} {#2}
    }
```

#1: char code; #2 char code or char range in Hex form. result: -1 if #1 before #2; 1 if #1 after #2; 0 otherwise.

```
  \cs_new_nopar:Npn \__interchar_class_compare_char:nn #1#2
    {
      \__interchar_class_compare_char_aux:www #1 - #2 - - \q_stop
    }
  \cs_new_nopar:Npn \__interchar_class_compare_char_aux:www #1 - #2 - #3 -
    {
      \tl_if_empty:nTF { #3 }
        {
          \int_compare:nTF { #1 = "#2 }
            { 0 } { \int_compare:nTF { #1 < "#2 } { -1 } { 1 } }
        }
        {
          \int_compare:nTF { "#2 <= #1 <= "#3 }
            { 0 } { \int_compare:nTF { #1 < "#2 } { -1 } { 1 } }
        }
      \__interchar_class_compare_char_stop:w
    }
  \cs_new_nopar:Npn \__interchar_class_compare_char_stop:w #1 \q_stop {}
```

#1: scheme name; #2: class number; #3 char range.

```
\cs_new_protected_nopar:Npn \__interchar_class_insert_char:nnn #1#2#3
  {
    % store all char ranges before #3
    \clist_clear:N \l_tmpa_clist
    % store all char ranges after #3
    \clist_set_eq:Nc
        \l_tmpb_clist { l_interchar_#1_chars_ \int_to_arabic:n{#2} _clist }
    \__interchar_class_split_range:nNN {#3} \l__interchar_a_tl \l__interchar_b_tl
    \tl_set:Nx \l_tmpa_tl { \int_to_hexadecimal:V \l__interchar_a_tl }
    \tl_set:Nx \l_tmpb_tl { \int_to_hexadecimal:V \l__interchar_b_tl }
    % if correct position found
    \bool_set_false:N \l_tmpa_bool
    \bool_do_until:Nn \l_tmpa_bool
      {
        \tl_if_empty:NTF \l_tmpb_clist
          { \bool_set_true:N \l_tmpa_bool }
          {
            \clist_pop:NN \l_tmpb_clist \l__interchar_xy_tl
            \exp_args:NV \__interchar_class_split_range:nNN
                { \l__interchar_xy_tl } \l__interchar_x_tl \l__interchar_y_tl
            \int_compare:nTF { "\l__interchar_y_tl < "\l_tmpa_tl - 1 }
              {
                % left
                \clist_put_right:NV \l_tmpa_clist \l__interchar_xy_tl
              }
              {
                \int_compare:nTF { "\l__interchar_x_tl > "\l_tmpb_tl + 1}
                  {
                    % right
                    \clist_put_left:NV \l_tmpb_clist \l__interchar_xy_tl
                    \bool_set_true:N \l_tmpa_bool
                  }
                  {
                    % middle
                    \int_compare:nT { "\l__interchar_x_tl < "\l_tmpa_tl }
                        { \tl_set_eq:NN \l_tmpa_tl \l__interchar_x_tl }
                    \int_compare:nT { "\l__interchar_y_tl > "\l_tmpb_tl }
                        { \tl_set_eq:NN \l_tmpb_tl \l__interchar_y_tl }
                  }
              }
          }
      }
    \tl_if_eq:NNTF \l_tmpa_tl \l_tmpb_tl
      { \tl_set_eq:NN \l__interchar_ab_tl \l_tmpa_tl }
      { \tl_set:Nx \l__interchar_ab_tl { \l_tmpa_tl - \l_tmpb_tl } }
    \clist_put_right:NV \l_tmpa_clist \l__interchar_ab_tl
    \clist_concat:cNN { l_interchar_#1_chars_ \int_to_arabic:n{#2} _clist }
                      \l_tmpa_clist \l_tmpb_clist
  }
```

#1: scheme name; #2: class number; #3: char range.

```
\cs_new_protected_nopar:Npn \__interchar_class_delete_char:nnn #1#2#3
  {
    % store all char ranges before #3
    \clist_clear:N \l_tmpa_clist
```

```
      % store all char ranges after #3
      \clist_set_eq:Nc
          \l_tmpb_clist { l_interchar_#1_chars_ \int_to_arabic:n{#2} _clist }
      \__interchar_class_split_range:nNN {#3} \l__interchar_a_tl \l__interchar_b_tl
      \tl_set:Nx \l_tmpa_tl { \int_to_hexadecimal:V \l__interchar_a_tl }
      \tl_set:Nx \l_tmpb_tl { \int_to_hexadecimal:V \l__interchar_b_tl }
      % if correct position found
      \bool_set_false:N \l_tmpa_bool
      \bool_do_until:Nn \l_tmpa_bool
        {
          \tl_if_empty:NTF \l_tmpb_clist
            { \bool_set_true:N \l_tmpa_bool }
            {
              \clist_pop:NN \l_tmpb_clist \l__interchar_xy_tl
              \exp_args:NV \__interchar_class_split_range:nNN
                  { \l__interchar_xy_tl } \l__interchar_x_tl \l__interchar_y_tl
              \int_compare:nTF { "\l__interchar_y_tl < "\l_tmpa_tl}
                {
                  % left
                  \clist_put_right:NV \l_tmpa_clist \l__interchar_xy_tl
                }
                {
                  \int_compare:nTF { "\l__interchar_x_tl > "\l_tmpb_tl}
                    {
                      % right
                      \clist_put_left:NV \l_tmpb_clist \l__interchar_xy_tl
                      \bool_set_true:N \l_tmpa_bool
                    }
                    {
                      % middle: put [x,a-1] and [b+1,y] into clist
                      \int_compare:nTF { "\l_tmpa_tl - "\l__interchar_x_tl = 1 }
                        {
                          \clist_put_right:NV \l_tmpa_clist \l__interchar_x_tl
                        }
                        {
                          \int_compare:nT { "\l_tmpa_tl - "\l__interchar_x_tl > 1 }
                          {
                            \tl_set:Nx \l__interchar_z_tl
                                { \int_to_hexadecimal:n { "\l_tmpa_tl - 1 } }
                            \clist_put_right:Nx \l_tmpa_clist
                                { \l__interchar_x_tl - \l__interchar_z_tl }
                          }
                        }
                      \int_compare:nTF { "\l__interchar_y_tl - "\l_tmpb_tl = 1 }
                        {
                          \clist_put_right:NV \l_tmpa_clist \l__interchar_y_tl
                        }
                        {
                          \int_compare:nT { "\l__interchar_y_tl - "\l_tmpb_tl > 1 }
                          {
                            \tl_set:Nx \l__interchar_z_tl
                                { \int_to_hexadecimal:n { "\l_tmpb_tl + 1 } }
                            \clist_put_right:Nx \l_tmpa_clist
                                { \l__interchar_z_tl - \l__interchar_y_tl }
                          }
```

```
                        }
                      \tl_if_eq:NNT \l_tmpa_tl \l_tmpb_tl
                          { \bool_set_true:N \g__interchar_class_delete_char_bool }
                    }
                }
            }
        }
      \clist_concat:cNN { l_interchar_#1_chars_ \int_to_arabic:n{#2} _clist }
                        \l_tmpa_clist \l_tmpb_clist
    }
```

Split #1 with - and put the results into #2 and #3.

```
  \NewDocumentCommand \__interchar_class_split_range:nNN
    { > { \SplitArgument { 1 } { - } } m m m}
    {
      \tl_set:No #2 { \use_i:nn #1}
      \tl_set:No #3 { \use_ii:nn #1}
      \exp_args:No \IfNoValueT {#3} { \tl_set_eq:NN #3 #2 }
    }
```

High level `\interchartoks` command. #1: scheme name; #2 and #3: class numbers; #4: tokens.

```
  \NewDocumentCommand \interchartoks { O{default} m m +m }
    {
      \interchar_toks:nnnn {#1} {#2} {#3} {#4}
    }
  \cs_new_protected_nopar:Npn \interchar_toks:nnnn #1#2#3#4
    {
      \int_set:Nn \l_tmpa_int {#2}
      \int_set:Nn \l_tmpb_int {#3}
      \prop_put:cxn { l_interchar_#1_toks_prop }
          { \int_use:N \l_tmpa_int \c_space_tl \int_use:N \l_tmpb_int }
          { #4 }
      \tl_if_eq:VoT \l_interchar_current_scheme_tl { #1 }
        {
          \xetex_interchartoks:D \l_tmpa_int \l_tmpb_int = { #4 }
        }
    }
  \cs_generate_variant:Nn \interchar_toks:nnnn { VVVV }
```

#1: scheme name; #2 and #3: class numbers; #4: result control sequence.

```
  \DeclareDocumentCommand \getinterchartoks { O{default} m m m }
    {
      \interchar_get_toks:nnnN {#1} {#2} {#3} {#4}
    }
  \cs_new_protected_nopar:Npn \interchar_get_toks:nnnN #1#2#3#4
    {
      \int_set:Nn \l_tmpa_int {#2}
      \int_set:Nn \l_tmpb_int {#3}
      \prop_get:cxN { l_interchar_#1_toks_prop }
          { \int_use:N \l_tmpa_int \c_space_tl \int_use:N \l_tmpb_int } #4
      \quark_if_no_value:NT #4 { \tl_clear:N #4 }
    }
```

`#1`: scheme name; `#2` and `#3`: class numbers. This function is fully expandable. If `#2` or `#3` is a control sequence generated from `\newintercharclass`, use the following function variants instead.

```
\cs_new_nopar:Npn \interchar_get_toks:nnn #1#2#3
  {
    \prop_item:cn { l_interchar_#1_toks_prop } { #2 ~ #3 }
  }
\cs_generate_variant:Nn \interchar_get_toks:nnn { nVn, nnV, nVV, VVV }
\xetex_intercharstate:D = 1
\ExplSyntaxOff
```

We need to call LaTeX3 functions.

```
\catcode `\_ = 11 \catcode `\: = 11
```

From now on, we use `\newcommand` for commands, `\def` for variables.

First we define some variables.

```
\def\interchar@scheme@name@tl{}
\def\interchar@c@tl{}
\def\interchar@tmpa@int{}
\def\interchar@tmpb@int{}
\def\interchar@tmpa@tl{}
```

Switch between getting and setting values.

```
\newif\ifinterchar@get
```

Used for migrating from XeTeX's primitive commands.

```
\let \NewIntercharScheme = \newintercharscheme
```

`#1`: scheme name.

```
\newcommand\interchar@migration[1]{%
  \expandafter\def\expandafter\interchar@c@tl\expandafter{\tl_upper_case:n #1}%
  \expandafter\newcommand\csname\interchar@c@tl IntercharState\endcsname{%
    \def\interchar@scheme@name@tl{#1}%
    \interchar@state@auxi
  }%
  \expandafter\newcommand\csname Get\interchar@c@tl IntercharState\endcsname{%
    \interchar_get_state:n{#1}%
  }%
  \expandafter\newcommand\csname New\interchar@c@tl IntercharClass\endcsname[1]{%
    \interchar_newclass:nn {#1} {##1}%
  }%
  \expandafter\newcommand\csname\interchar@c@tl IntercharClass\endcsname{%
    \def\interchar@scheme@name@tl{#1}%
    \interchar@class@auxi
  }%
  \expandafter\newcommand\csname Get\interchar@c@tl IntercharClass\endcsname{%
    \def\interchar@scheme@name@tl{#1}%
    \interchar@gettrue
    \interchar@class@auxi
  }%
  \expandafter\newcommand\csname\interchar@c@tl IntercharToks\endcsname{%
    \def\interchar@scheme@name@tl{#1}%
```

```
      \interchar@toks@auxi
    }%
    \expandafter\newcommand\csname Get\interchar@c@tl IntercharToks\endcsname{%
      \def\interchar@scheme@name@tl{#1}%
      \interchar@gettrue
      \interchar@toks@auxi
    }%
  }
```

Commands for scanning number or toks arguments.

```
  \newcommand\interchar@scan@number[1]{%
    \afterassignment#1\count255 %
  }
  \newcommand\interchar@scan@number@x[1]{%
    \afterassignment#1\count255=%
  }
  \newcommand\interchar@scan@toks[1]{%
    \afterassignment#1\toks0 %
  }
```

Scanning arguments of `\FOOinterchartokenstate` command.

```
  \newcommand\interchar@state@auxi{%
    \interchar@scan@number \interchar@state@auxii
  }
  \newcommand\interchar@state@auxii{%
    \edef\interchar@tmpa@int{\the\count255}%
    \interchar_state:VV \interchar@scheme@name@tl \interchar@tmpa@int
  }
```

Scanning arguments of `\FOOcharclass` command.

```
  \newcommand\interchar@class@auxi{%
    \interchar@scan@number@x \interchar@class@auxii
  }
  \newcommand\interchar@class@auxii{%
    \edef\interchar@tmpa@int{\the\count255}%
    \ifinterchar@get
      \interchar@getfalse
      \interchar_get_class:VV \interchar@scheme@name@tl \interchar@tmpa@int
    \else
      \interchar@scan@number \interchar@class@auxiii
    \fi
  }
  \newcommand\interchar@class@auxiii{%
    \edef\interchar@tmpb@int{\the\count255}%
    \interchar_class:VVV \interchar@scheme@name@tl
            \interchar@tmpa@int \interchar@tmpb@int
  }
```

Scanning arguments of `\FOOinterchartoks` and `\getFOOinterchartoks` command.

```
  \newcommand\interchar@toks@auxi{%
    \interchar@scan@number@x \interchar@toks@auxii
  }
  \newcommand\interchar@toks@auxii{%
    \edef\interchar@tmpa@int{\the\count255}%
```

```
    \interchar@scan@number@x \interchar@toks@auxiii
  }
  \newcommand\interchar@toks@auxiii{%
    \edef\interchar@tmpb@int{\the\count255}%
    \ifinterchar@get
      \interchar@getfalse
      \interchar_get_toks:VVV \interchar@scheme@name@tl
          \interchar@tmpa@int \interchar@tmpb@int
    \else
      \interchar@scan@toks \interchar@toks@auxiv
    \fi
  }
  \newcommand\interchar@toks@auxiv{%
    \edef\interchar@tmpa@tl{\the\toks0}%
    \interchar_toks:VVVV \interchar@scheme@name@tl
            \interchar@tmpa@int \interchar@tmpb@int \interchar@tmpa@tl
  }
```

Recover catcode changes.

```
  \catcode `\_ = 8 \catcode `\: = 12
```