# CREPDL: Protect Yourself from the Proliferation of Unicode Characters

Makoto Murata, Keio University and JEPA <eb2mmrt@gmail.com>

**Abstract**

This paper studies machine-readable notations for describing subsets of Unicode or ISO/IEC 10646. Unicode regular expressions can describe any subset, but they have performance problems for huge subsets and cannot directly capture subsets defined in terms of other subsets. Meanwhile, the upcoming second edition of ISO/IEC 19757-7 Character Repertoire Description Language (CREPDL) overcomes these problems by providing references to well-known subsets and external CREPDL scripts.

## Table of Contents

Introduction ........ 1
Subsets in Unicode ........ 2
Subsets in ISO/IEC 10646 ........ 2
    Collections ........ 2
Existing Machine-readable Notations for Describing Subsets ........ 4
    Unicode Regular Expressions ........ 4
    A Notation for Character Collections for the WWW ........ 5
Design and Implementation of CREPDL ........ 5
    Language Design ........ 5
    Implementation ........ 6
Concluding Remarks and Future Works ........ 6
Bibliography ........ 6

## Introduction

Which character in Unicode are you willing to accept? If you receive UTF-8 text from somebody, it might contain any of the 136,690 code points of Unicode 10.

Accepting any Unicode character may lead to problems in the future. First, nobody can read all characters. Second, few fonts cover all characters. Third, some software such as document editors supports only a subset of Unicode.

Historically, legacy encodings have protected users from the proliferation of characters. For example, as long as you use Shift JIS, you only have to worry about 7,000 characters. But UTF-8 now exposes almost 88,000 CJK ideographic characters.

In the Unicode era, we need a language for describing which character is to be allowed and then examining text against descriptions in this language. ISO/IEC 19757-7 Character REPertoire Description Language (CREPDL) [3] is an attempt of ISO/IEC JTC1/SC34 for such a language. Although the first edition was restricted to code points, the second edition can handle code point sequences, which represent grapheme clusters ("user-perceived characters").

The rest of this paper is organized as follows. In Sections 2 and 3, we study subsets in Unicode and ISO/IEC 10646, respectively. In Section 4, we study two existing machine-readable notations for describing subsets: regular expressions [7] and a W3C notation [9]. In particular, we make clear that regular expressions have performance problems for huge subsets and cannot directly capture subsets defined in terms of other subsets. In Section 5, we have a quick overview of the design and implementation of CREPDL and see how it overcomes limitations of the existing notations.

## Subsets in Unicode

The Unicode standard [5] does not mandate the support of all Unicode characters. Rather, it allows implementations to support subsets of Unicode characters.

But Unicode does not define any subsets. It does not provide any mechanisms for specifying such subsets either. This is made clear by the following bullet extracted from "Interpretation" subsubclausse in the subclause 3.2 (Conformance Requirements) of the Unicode standard.

- Any means for specifying a subset of characters that a process can interpret is outside the scope of this standard.

However, it is true that Unicode regular expressions can be used for representing subsets. We will discuss this topic in Section 4.1.

## Subsets in ISO/IEC 10646

Likewise, conformant implementations of ISO/IEC 10646 [2] may support subsets rather than all characters in ISO/IEC 10646.

But ISO/IEC 10646 goes much further than Unicode. A subset is defined as either an implementation-defined list of code points, a standardized collection, or a combination of the two. Collections are standardized in Annex A of ISO/IEC 10646.

In the following subsections, we study collections in Annex A. But it is interesting to note that ISO/IEC 10646 does not provide a standard notation for specifying subsets. Different collections in Annex A are represented by different notations.

## Collections

### Code Points and Ranges

Most collections in Annex A are very simple. They are ranges of code points. For example, `LATIN-1 SUPPLEMENT` (collection 2) is a range 00A0-00FF.

`MULTILINGUAL EUROPEAN SUBSET 2` (collection 282) is more complicated. It is specified by the following ranges of code points as indicated for each row.

**Example 1. `MULTILINGUAL EUROPEAN SUBSET 2`**

```
Plane 00
Row Values within row
00  20-7E A0-FF
01  00-7F 8F 92 B7 DE-EF FA-FF
02  18-1B 1E-1F 59 7C 92 BB-BD
    C6-C7 C9 D8-DD EE
03  74-75 7A 7E 84-8A 8C 8E-A1
    A3-CE D7 DA-E1
04  00-5F 90-C4 C7-C8 CB-CC D0-EB
    EE-F5 F8-F9
1E  02-03 0A-0B 1E-1F 40-41 56-57
    60-61 6A-6B 80-85 9B F2-F3
```

```
1F  00-15 18-1D 20-45 48-4D 50-57
    59 5B 5D 5F-7D 80-B4 B6-C4
    C6-D3 D6-DB DD-EF F2-F4 F6-FE
20  13-15 17-1E 20-22 26 30 32-33
    39-3A 3C 3E 44 4A 7F 82 A3-A4
    A7 AC AF
21  05 16 22 26 5B-5E 90-95 A8
22  00 02-03 06 08-09 0F 11-12 19-1A
    1E-1F 27-2B 48 59 60-61 64-65
    82-83 95 97
23  02 10 20-21 29-2A
25  00 02 0C 10 14 18 1C 24 2C 34 3C
    50-6C 80 84 88 8C 90-93
    A0 AC B2 BA BC C4 CA-CB D8-D9
26  3A-3C 40 42 60 63 65-66 6A-6B
    FB 01-02
FF  FD
```

Although this collection is not small, CJK collections are significantly larger. For example, `JIS2004 IDEOGRAPHICS EXTENSION` (collection 371) has 3695 code points. `BASIC JAPANESE` (collection 285) contains 6884 code points. IICORE (collection 370) has 9810 code points. Ranges are not useful for such CJK collections since code points in them are scattered. The definitions of these CJK collections are provided as electronic attachments.

## Open Collections and Fixed Collections

Some collections defined in Annex A contain unassigned code points. Such collections are called *open* collections. Meanwhile, *fixed* collections contain assigned code points only.

Unassigned code points in open collections may be assigned by later versions of ISO/IEC 10646. Meanwhile, fixed collections remain unchanged in future versions.

## References to Other Collections

Some collections are defined as the union of other collections. For example, `MODERN EUROPEAN SCRIPTS` (collection 283) is the union of more than 30 collections, each of which is a simple range. `COMMON JAPANESE` (collection 287) is defined as the union of `BASIC JAPANESE` (collection 285) and an enumerated list of 609 code points. Although it is possible to copy the definition of referenced collections and avoid references, the result would be less readable and harder to maintain.

## Grapheme Clusters

A grapheme cluster [6] is a sequence of code points that represents“user-perceived characters”. A simple example is a base character followed by a combining character.

`CONTEMPORARY LITHUANIAN LETTERS` (collection 284) is the first collection containing grapheme clusters such as <004A, 0303> and <0069, 0307, 0301>. Note that 0303 is allowed to follow some code points (e.g, 004A), but is not allowed to follow others (e.g., 004B).

`MOJI-JOHO-KIBAN IDEOGRAPHS-2016` (collection 390) is a collection applicable to persons' names in Japanese public service. This collection contains grapheme clusters such as <5289,E0101> and <5351,FE00>, where E0101 is an ideographic variation selector and FE00 is a variation selector. Although E0101 is allowed to follow 5289, it is not allowed to follow other characters (5288, for example).

The size of `CONTEMPORARY LITHUANIAN LETTERS` is much smaller than that of `MOJI-JOHO-KIBAN IDEOGRAPHS-2016`. The number of code points and grapheme clusters in `CONTEMPORA BY LITHUANIAN LETTERS` (collection 284) is less than 100. But the number of code points in `MOJI-JOHO-KIBAN IDEOGRAPHS-2016` is more than 52000 and that of grapheme clusters is more than 10000.

## User-defined Subsets

### Subsets Defined by Governments

Although collections in ISO/IEC 10646 are defined for technical reasons, the Japanese government defines sets of CJK ideographic characters for non-technical reasons. For example, Kyouiku Kanji [4]is a set of CJK ideographic characters for elementary school education. It has 1006 characters. Another set, Jouyou Kanji, contains 2136 CJK ideographic characters for use in official government documents. Governments in Mainland China, Taiwan, Hong-Kong, and Korea also define sets of CJK ideographic characters.

### Subsets for Describing Font Coverage

Characters covered by commercial fonts in Japan are slightly different from ISO/IEC 10646 collections for some reasons (backward compatibilities with Shift JIS, for example). Machine-readable descriptions of font-covered subsets make it easier to compare different fonts having different coverage.

# Existing Machine-readable Notations for Describing Subsets

## Unicode Regular Expressions

Unicode regular expressions [7] can be used for representing Unicode subsets. In fact, in the Unicode Common Locale Data Repository [1], subsets for each locale are represented by regular expressions. Subsubsection 5.3.3 (Unicode Sets) of Unicode Technical Standard #35 [8] describes the use of regular expressions for subsets and demonstrates the use of code points, ranges, code point sequences, and set operations (union, inverse, difference, and intersection).

Any collection defined in ISO/IEC 10646 can be represented by a Unicode regular expressions. In particular, code point sequences representing grapheme clusters (e.g., `<5289,E0101>`) can be represented by regular expressions (e.g., `{\u5289\U000E0101}`).

However, there are two problems. When collections are small, these problems are insignificant. But they become quite significant for large collections such as CJK collections.

### Referencing other subsets

Unicode regular expressions cannot reference collections defined in ISO/IEC 10646. Likewise, they cannot reference other regular expressions. Thus, regular expressions cannot directly capture collections defined in terms of other collections. It is thus necessary to create a gigantic regular expression by copying the definition of each referenced collection. This might be acceptable for MODERN EUROPEAN SCRIPTS (collection 283). But it is too inconvenient for COMMON JAPANESE (collection 287) since it references JIS2004 IDEOGRAPHICS EXTENSION (collection 371), which contains 3695 code points.

### Performance

Unicode regular expression engines are slow for large collections. However, hash-based set operations are much faster. This observation is based on an experiment with ICU's Regular Expressions package and a hash-based set of the programming language F#.

First, I created a regular expression for the MULTILINGUAL EUROPEAN SUBSET 2 collection. After creating a matcher (an instance of the class RegexMatcher) from it, I invoked it for the string

"&#x3000;" 100000 times. In my computing environment (AMD A10-7800, 16GB Memory, Windows 10), the elapsed time was about 0.4 seconds.

I then created a hash-based set for the same collection and tested if it contains the same string. I did this test 100000 times. The elapsed time was about 0.015 seconds. Thus, the hash-based set is more than 20 times faster than the ICU's Regular Expressions package.

Second, I did the same experiment for the IICORE collection. In the case of the regular expression matcher, the elapsed time was about 23 seconds. In the case of the hash-based set, the elapsed time was about 0.015 seconds. Thus, the hash-based set is more than 1600 times faster than the ICU's Regular Expressions package.

The slow performance of regular expression engines might not be problematic if the collection is not large. But it is fatal for huge CJK collections.

## A Notation for Character Collections for the WWW

“A Notation for Character Collections for the WWW” [9] (hereafter W3C notation for short) provides an XML syntax for describing subsets. Although it has not become a W3C recommendation and has not been implemented, it has a number of interesting ideas.

The W3C notation does not use regular expressions. Rather, it introduces XML elements (`range` and `enum`) for representing ranges and code points, respectively.

An interesting feature of the W3C notation is its `kernel` and `hull` elements. They are used to define open collections.

Unlike regular expressions, the W3C notation is equipped with a mechanism that references other subset descriptions or well-known subsets (e.g., collections in ISO/IEC 10646). This notation can thus easily describe subsets defined in terms of other subsets.

The W3C notation also has set operations (union, inverse, difference, and intersection). They allow subsets to be defined in terms of other subsets.

However, the W3C notation lacks mechanisms for describing grapheme clusters.

# Design and Implementation of CREPDL

## Language Design

CREPDL is intended to combine the best parts of Unicode regular expressions and the W3C notation. Unlike regular expressions, CREPDL can easily handle large collections. Unlike the W3C notation, CREPDL can handle grapheme clusters.

First, CREPDL allows the use of Unicode regular expressions as atomic expressions. This is done by the `char` element of CREPDL. Note that sequences of code points, which represent grapheme clusters, can be represented by regular expressions.

Second, CREPDL borrows mechanisms of the W3C notation with some modifications.

- CREPDL allows references to collections defined in ISO/IEC 10646 and other well-known subsets. The `repertoire` element of CREPDL represents such references. For example, IICORE (collection 370) can be referenced by `<repertoire registry="10646" number="370"/>`.
- CREPDL allows references to other CREPDL scripts by URIs. The `ref` element of CREPDL represents such references.
- CREPDL provides set operation by the `union`, `intersection`, and `difference` elements.

- CREPDL allows open collections and fixed collections by the `kernel` and `hull` elements.

The CREPDL processor has two working modes: `character` and `graphmeCluster`. If the mode is `character`, the CREPDL processor examines each code point in the input text stream. If the mode is `graphemeCluster`, the CREPDL processor extracts grapheme clusters from the text stream by applying the algorithm as defined in [6]. It then validates each grapheme cluster.

Huge well-known collections referenced by `<repertoire` can be implemented by hash-based sets. Thus, the CREPDL processor can handle such collections very efficiently.

This paper does not cover details of the CREPDL language. Interested readers are encouraged to review the CD or upcoming DIS for ISO/IEC 19757-7.

## Implementation

An open source implementation of CREPDL is available at `https://github.com/CIT-PCSHARE/CREPDL`. It is written in F# (a functional programming language). This implementation relies on the ICU regular expression engine.

Large collections in Annex A of ISO/IEC 10646 are implemented as hash-based sets. Validation against such collections is thus very efficient.

Another GitHub repository provides a collection of example CREPDL scripts. It is available at `https://github.com/CITPCSHARE/CREPDLScripts`.

# Concluding Remarks and Future Works

The upcoming revision of CREPDL is intended to combine the best parts of Unicode regular expressions and the W3C notation. CREPDL is expected to work nicely for huge subsets. The F# implementation of CREPDL is publicly available.

The next step is to use CREPDL for comparing different subsets covered by different fonts. It is not easy to directly compare two CREPDL scripts. However, if there is a list of all grapheme clusters in Unicode, validation of this list against two CREPDL scripts can provide definitive answers.

# Bibliography

[1] CLDR - Unicode Common Locale Data Repository http://cldr.unicode.org/

[2] *ISO/IEC 10646, Universal Coded Character Set (UCS)* .

[3] *ISO/IEC CD 19757-7:2017, Document Schema Definition Languages (DSDL) — Part 7: Character Repertoire Description Language (CREPDL)*.

[4] Kyōiku kanji, Wikipedia https://en.wikipedia.org/wiki/Kyōiku_kanji

[5] The Unicode Consortium. The Unicode Standard. http://www.unicode.org/versions/latest/

[6] Unicode Standard Annex #29: Unicode Text Segmentation http://www.unicode.org/reports/tr29/

[7] Unicode Technical Standard #18: Unicode Regular Expressions http://www.unicode.org/reports/tr18/

[8] Unicode Technical Standard #35: Unicode Locale Data Markup Language (LDML) http://www.unicode.org/reports/tr35/

[9] W3C Note 14-January-2000: A Notation for Character Collections for the WWW https://www.w3.org/TR/2000/NOTE-charcol-20000114/