Technical Reports

L2/07-362

Proposed Draft Unicode Technical Report #42

AN XML REPRESENTATION OF THE UCD

Author	Eric Muller (emuller@adobe.com)
Date	2007-04-21
This Version	http://www.unicode.org/reports/tr42/tr42-1.html
Previous Version	n/a
Latest Version	http://www.unicode.org/reports/tr42/
Schema	http://www.unicode.org/reports/tr42/tr42-1.rnc
Tracking Number	1

Summary

This document describes an XML representation of the Unicode Character Database.

Status

*This is a **draft** document which may be updated, replaced, or superseded by other documents at any time. Publication does not imply endorsement by the Unicode Consortium. This is not a stable document; it is inappropriate to cite this document as other than a work in progress.*

> ***A Unicode Technical Report (UTR)** contains informative material. Conformance to the Unicode Standard does not imply conformance to any UTR. Other specifications, however, are free to make normative references to a UTR.*

Please submit corrigenda and other comments with the online reporting form [Feedback]. Related information that is useful in understanding this document is found in References. For the latest version of the Unicode Standard see [Unicode]. For a list of current Unicode Technical Reports see [Reports]. For more information about versions of the Unicode Standard, see [Versions].

Contents

1 Introduction

In working on Unicode implementations, it is often useful to access the full content of the Unicode Character Database (UCD). For example, in establishing mappings from characters to glyphs in fonts, it is convenient to see the character scalar value, the character name, the character east asian width, along with the shape and metrics of the proposed glyph to map to; looking at all this data simultaneously helps in evaluating the mapping.

Accessing directly the data files that constitute the UCD is sometime a daunting proposition. The data is dispersed in a number of files of various formats, and there are just enough peculiarities (all justified by the processing power available at the time the UCD representation was designed) to require a fairly intimate knowledge of the data format itself, in addition to the meaning of the data.

Many programming environments (e.g. Java or ICU) do give access to the UCD. However, those environments tend to lag behind releases of the standard, or support only some of the UCD content.

Unibook is a wonderful tool to explore the UCD and in many cases is just the ticket; however, it is difficult to use when the task at hand has not been built-in, or when non-UCD data is to be displayed along.

This paper presents an alternative representation of the UCD, which is meant to overcome these difficulties. We have chosen an XML representation, because parsing becomes a non-issue: there are a number of XML parsers freely available, and using them is often fairly easy. In addition, there are freely available tools that can perform powerful operations on XML data; for example, XPATH and XQUERY engines can be thought of a “grep” for XML data and XSLT engines can be thought of as “awk” for XML data.

It is important to note that we are interested in exploring the content of the UCD, rather than using the UCD data in processing to character streams. Thus, we are not concerned so much by the speed of processing or the size of our representation.

Our representation supports the creation of documents that represent only parts of the UCD, either by not representing all the characters, or by not representing all the properties. This can be useful when only some of the data is needed.

2 Overall schema

2.1 General principles

Our schema defines a set of valid documents which are intended to represent properties of Unicode code points, blocks, named sequences, normalization corrections, and standardized variants. A document may represent the values actually assigned in a given version of the UCD, or it may represent a draft version of the UCD, or a private agreement on Private Use characters. The validity of a document does not assert anything about the correctness of the values.

Valid documents may provide values for only some of the the code points, or some of the Unicode properties. Furthermore, they may also incorporate non-Unicode properties.

Our schema is defined using English. However, a useful subset of the validity constraints can be captured using a schema language, thereby simplifying the task of validating documents. We have chosen Relax NG *[ISO 19757]*, in the compact syntax *[ISO 19757 Amd1]*, as the schema language. It is important to stress that the Relax NG schema does not define valid documents.

While our XML representation is not intented to be used during processing of characters and strings, it is still a design principle for our schema to supports the relatively efficient representation of the UCD. This is achieved by an inheritance mechanism, similar to property inheritance in CSS or in XSL:FO.

2.2 Namespace

The namespace for our elements is “http://www.unicode.org/ns/2003/ucd/1.0”. Our attributes are in the empty namespace.

[namespace declaration, 1] = `default namespace ucd = "http://www.unicode.org/ns/2003/ucd/1.0"`

In all our examples, we assume that this namespace is the default one.

Non-Unicode properties can be represented by using elements in another (possibly empty) namespace, and/or by attributes in a non-empty namespace. Such elements are ignored for the purpose of determining the validity of a document. The namespace above (for elements) and the empty namespace (for attributes) are reserved for use by this specification.

2.3 Datatypes

We use a standard XML Schema datatypes:

[datatypes declaration, 2] = `# default; datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"`

Characters are pervasive in the UCD, and will need to be represented. Representing characters directly by themselves would seem the most obvious choice; for example, we could express that the decomposition of U+00E8 is “&#x0065;&#x0300;”, i.e. have exactly two characters in (the infoset of) the XML document. However, the current XML specification limits the set of characters that can be part of a document. Another problem is that the various tools (XML parser, XPATH engine, etc.) may equate U+00E8 with U+0065 U+0300, thus making it difficult to figure out which of the two sequences is contained in the database (which is sometimes important for our purposes). Therefore, we chose instead to represent characters by their code points; we follow the usual convention of four to six hexadecimal digits (uppercase) and code points in a sequence separated by space; e.g., the decomposition of U+00E8 will be represented by the nine characters “0065 0300” in the infoset.

[datatype for code points, 3] = `single-code-point = xsd:string { pattern = "(|[1-9A-F]|(10))[0-9A-F]{4}" }`

```
  one-or-more-code-points = list { single-code-point + }
  zero-or-more-code-points = list { single-code-point * }
```

2.4 Root Element

The root element of valid documents is a `ucd`.

[schema start, 4] = `start =`
```
    element ucd { ucd.content }
```

2.5 Common attributes

A large number of properties are boolean. We uniformly use the values `Y` and `N` for those:

[boolean type, 5] =
```
boolean = "Y" | "N"
```

3 Description

The root element may have a `description` child element, which is turn contains any string, which is meant to describe what the XML document purports to describe.

It is recommended that if the document purports to represent the UCD of some Unicode version, the `description` be selected in accord with the rules listed in ??; and conversely, that documents which do not purport to represent the UCD be described as such.

[description, 6] =
```
ucd.content &=
    element description { text }?
```

4 Repertoire

The `repertoire` child element of the `ucd` element describes the code points and their properties. As we will see shortly, code points can be described individually or as part of a group:

[repertoire, 7] =
```
ucd.content &=
    element repertoire { (code-point | group)* }?
```

4.1 Sets of code points

It is often the case that successive code points have the the same property values, for a given set of properties. The most striking example is that of an unallocated plane, where all but the last two code points are reserved and have the same property values. Another example is the URO (U+4E00 .. U+9FA5) where all the code points have the same property values if we ignore their name and their Unihan properties.

This observation suggests that it is profitable to represent sets of code points which share the same properties, rather than individual code points. To make the representation of the sets simple, we restrict them to be segments in the code point space, i.e. a set is defined by the first and last code point it contains. Those are captured by the attributes `first-cp` and `last-cp`. The attribute `cp` is a shorthand notation for the case where the set has a single code point.

[Set of code points, 8] =
```
set-of-code-points =
     attribute cp { single-code-point }
   | ( attribute first-cp { single-code-point },
       attribute last-cp  { single-code-point } )
```

While we represent *sets* of code point, the corresponding element is nevertheless named `code-point`:

[code point, 9] =
```
code-point |=
    element code-point {
      set-of-code-points,
      code-point.content }
```

In the `repertoire`, there must be at most one `code-point` element for a given code point.

4.2 Code point types

When thinking about Unicode code points, it is useful to split them into four types:

- those assigned to abstract characters (PUA or not)
- the non characters
- the surrogate code points
- the reserved code points

We capture that characterization using the `type` attribute:

[code points, 10] =
```
code-point.content =
    attribute type { "reserved" | "noncharacter" | "surrogate" | "char" }?,
    code-point-properties
```

We have shorthand notation to combine the `code-point` element name and the `type` attribute:

[shorthand for code-point + type, 11] =
```
code-point |=
    element reserved {
      set-of-code-points,
      code-point-properties }

  code-point |=
    element noncharacter {
      set-of-code-points,
      code-point-properties }

  code-point |=
    element surrogate {
      set-of-code-points,
      code-point-properties }
  code-point |=
    element char {
      set-of-code-points,
      code-point-properties }
```

4.3 Group

While we already recognized the situation where a set of code points have exactly the same set of property values, another common situation is that of code points which have almost all the same property values.

For example, the characters U+1740 BUHID LETTER A .. U+1753 BUHID VOWEL SIGN U all have the age “3.2”, and all have the script “Buhd”. On the one hand, it is convenient to support data files in which those properties are explicitly listed with every code point, at this make answering questions like “what is the age of U+1749?” easier, since that data is expressed right there. On the other hand, this leads to rather large data files, and it also tends to obscure the differences between similar characters.

Our representation accounts for this situation with the notion of groups. A `group` element is simply a container of code points that also holds default values for the properties. If a code point inside a `group` does not list explicitly a property but the `group` lists it, then the code point inherits that property from its `group`. For example, the fragment with explicit properties:

```
  <char cp="1740" age="3.2" na="BUHID LETTER A" gc="Lo" sc="Buhd"/>
  <char cp="1741" age="3.2" na="BUHID LETTER I" gc="Lo" sc="Buhd"/>
  <char cp="1752" age="3.2" na="BUHID VOWEL SIGN I" gc="Mn" sc="Buhd"/>
  <char cp="1820" age="3.0" na="MONGOLIAN LETTER A" gc="Lo" sc="Mong"/>
```

is equivalent to this fragment which uses a `group`:

```
  <group age="3.2" gc="Lo" sc="Buhd">
    <char cp="1740" na="BUHID LETTER A"/>
    <char cp="1741" na="BUHID LETTER I"/>
    <char cp="1752" na="BUHID VOWEL SIGN I" gc="Mn"/>
    <char cp="1820" age="3.0" na="MONGOLIAN LETTER A" sc="Mong"/>
  </group>
```

The element for U+1740 does not have the `age` attribute, and it therefore inherits it from its enclosing `group` element, i.e. “3.2”. On the other hand, the element for U+1820 does have this attribute, so the value is “3.0”.

As this example illustrates, the notion of `group` does not necessarily align with the notion of Unicode block. It is entirely defined and limited to our representation. In particular, the value of a property for a code point can always be determined from the XML document alone, assuming that this property and this code point are expressed at all. Of course, one may create an XML representation where the groups happen to coincide with the Unicode blocks.

Groups cannot be nested. The motivation for this limitation is to make the life of consumers easier: either a property is defined by the element for a code point, or it is defined by the immediately enclosing `group` element.

[groups, 12] = `group =`

```
   code-point* }
```

4.4 Properties

d by an attribute.
Unihan properties, the name is that given in the various versions of Unihan.txt (some properties are no longer present in version 5.0.0).

in version 5.0.0).
is an abbreviated name, it is used, otherwise the long name is used.

long name is used.

4.1 Age property

or non-character.

```
.0" | "4.1" | "5.0" | "unassigned" }?

ssigned" }?
```
his character had in version 1.0 of the standard

in version 1.0 of the standard (`na1`).

```
ern="([A-Z0-9 #\-\(\)]*)|(<control>)" }

perties &= attribute na1 { character-name }?
```
e point>. It also happens that character names cannot contain the character U+0023 # NUMBER SIGN, so we adopt the following convention: if a code point has has the attribute `na` (either directly or by inheritence from an enclosing group), then occurrences of the character # in the name are to be interpreted as the

d as the value of the code point. For example:

and

```
<char cp="3400" na="CJK UNIFIED IDEOGRAPH-#"/>
```

are equivalent. The # can be in any position in the value of the `na` attribute. The convention also applies just as well to a set of multiple code points:

```
<char cp="3400" na="CJK UNIFIED IDEOGRAPH-3400"/>
<char cp="3401" na="CJK UNIFIED IDEOGRAPH-3401"/>
```

is equivalent to

```
<char cp="3400" na="CJK UNIFIED IDEOGRAPH-#"/>
<char cp="3401" na="CJK UNIFIED IDEOGRAPH-#"/>
```

which in turn is equivalent to:

```
<char first-cp="3400" last-cp="3401" na="CJK UNIFIED IDEOGRAPH-#"/>
```

4.4.3 General Category

The general category is represented by the `gc` attribute.

[gc property, 16] = `code-point-properties &=`

```
          | "Nd" | "Nl" | "No"

| "Sk" | "So"

| "Zs" | "Zl" | "Zp"
```

g class is represented by the `ccc` attribute, which holds the decimal representation of the combining class.
ken accross the various versions of the UCD is rather

versions of the UCD is rather large, our schema does not restrict the possible values to those actually used.

ty, 17] = `code-point-properties &=`

```
=

"255" }}?
nt-properties &=

               | "B " | "BN"

               | "L"  | "LRE" | "LRO"

                  | "PDF"
```

`Bidi_M` attribute, which takes a boolean value.

e.

If the mirrored property is true, then the `bmg` attribute is significant (if present). Its value is the code point of a character whose glyph is typically a mirrored
irrored image of the glyph for the current character.

```
oint }?
```

d in BidiMirroring.txt. For one thing, it is not meant to be machine readable. More importantly, the idea underlying the mirrored glyph is delicate to use, since it

ortantly, the idea underlying the mirrored glyph is delicate to use, since it

irrored glyph is delicate to use, since it makes assumptions about the design of the fonts, and the best fit goes even

oes even farther.

```
ttribute Bidi_C { boolean }?
```

`dm` attributes.

ion mapping to themselves. This is very similar to the situation we encountered with names, and we adopt a similar convention: if the value of a decomposition
ition mapping is the character itself, we use the attribute value # (U+0023 # NUMBER SIGN) as a shorthand notation; this enables those attributes to be

e captured in groups.

```
 "enc" | "fin"  | "font" | "fra"

vert" | "wide" | "none"}?
```

n_Exclusion and Full_Composition_Exclusion are represented by the attributes

he attributes `CE` and `Comp_Ex`:

```
es &=
```

4.4.7 Numeric Properties

The numeric type is represented by the `nt` attribute.

If the numeric type is not `None`, then the numeric value is represented by the `nv` attribute, represented as a fraction.

[numeric properties, 25] = `code-point-properties &=`

```
te nv { "" | xsd:string { pattern = "-?[0-9]+(/[0-

9]+)?" }}?
```

Properties

attribute.

e character.

```
rties &=

aw" | "Tah" | "Taw"

 "Yeh_Barree" | "Yeh_With_Tail" | "Yudh"
```

_Control property is representedy by the

dy by the `Join_C` attribute.

```
  code-point-properties &=
```

ibute.

```
" | "GL" | "H2" | "H3"

  | "HY" | "ID" | "IN" | "IS" | "JL" | "JT" | "JV"

| "NS" | "NU" | "OP" | "PO" | "PR"

| "ZW" }?
```

sian Width Properties

```
  code-point-properties &=
```

rcase and Other_Lowercase properties are represented by corresponding attributes.

es, 30] = `code-point-properties &=`

```
bute OUpper { boolean }? code-point-properties &= attribute OLower { boolean }?
```

mply map or fold to themselves. This is very similar to the situation we encountered with names, and we adopt a similar convention: if the value of a case

adopt a similar convention: if the value of a case mapping or case folding property the character itself, we use the attribute value # (U+0023 #

perty the character itself, we use the attribute value # (U+0023 # NUMBER SIGN) as a shorthand notation; this enables those attributes to be captured in groups.
ed in groups.

[casing properties, 31] = `code-point-properties &=`

```
uc { "#" | single-code-point }?
c { "#" | single-code-point }?
```

utes.

```
 }?
-points }?
```

le_Case_Folding and Case_Folding properties are recorded in the `sfc` and `cf` attributes respectively.

```
 &=

 single-code-point }?

bute cf { "#" | one-or-more-code-points }?
```

ically the entries in CaseFolding.txt with status T.

```
"Laoo" | "Latn" | "Limb" | "Linb" | "Mlym" | "Mong" | "Mymr" | "Nkoo" | "Ogam" | "Orya" | "Osma" | "Phag" | "Phnx" | "Qaai" | "Runr" | "Shaw" | "Sinh"

" | "Ogam" | "Orya" | "Osma" | "Phag" | "Phnx" | "Qaai" | "Runr" | "Shaw" | "Sinh" | "Sylo" | "Syrc" | "Tagb" | "Tale" | "Talu" | "Taml" | "Telu" | "T

inh" | "Sylo" | "Syrc" | "Tagb" | "Tale" | "Talu" | "Taml" | "Telu" | "Tfng" | "Tglg" | "Thaa" | "Thai" | "Tibt" | "Ugar" | "Xpeo" | "Xsux" | "Yiii" |
```

The property Hangul_Syllable_Type is represented by the `hst` attribute.

[hst property, 36] = `code-point-properties &=`

```
" }?
```

ibute:

```
]{0,3}" }}?
```

properties

g attributes:

```
DS { boolean }?

-point-properties &=
```

ite_Space are represented by corresponding attributes:

```
 }?
```

4.4.16 Properties related to function and graphic characteristics

ender, Soft_Dotted, Alphabetic, Other_Alphabetic, Math, Other_Math, Hex_Digit, ASCII_Hex_Digit, Default_Ignorable_Code_Point,
le_Code_Point, Logical_Order_Exception and White_Space describe the function or graphic characteristic of a character, and have each a corresponding

acteristic of a character, and have each a corresponding attribute.

```
nt-properties &=

boolean }?

ttribute ODI { boolean }?
```

7 Properties related to boundaries

es

each have a corresponding attribute:

```
T" | "V"

e-point-properties &=

operties &=
```

ted to ideographs

ies related to ideographs, 42] =

```
  code-point-properties &=

 { boolean }?
rties &=
```

4.4.19 Miscellaneous properties

rties

Code_Point have corresponding attributes:

```
-point-properties &=
} +}}? code-point-properties &= attribute kPrimaryNumeric { xsd:string {pattern="[0-9]+"} }? code-point-properties &= attribute kPseudoGB1 { xsd:strin

n="U\+2?[0-9A

-F]{4}"} +}}? code-point-properties &= attribute kVietnamese { list { xsd:string {pattern="[A-Za-zà-uạ-ỹ]+"} +}}? code-point-properties &= attribute k

 }? code-point-properties &= attribute kXerox { xsd:string {pattern="[0-9]{3}:[0-9]{3}"} }? code-point-properties &= attribute kZVariant { xsd:string
pattern="U\+2?[0

="U\+2?[0-9A-F]{4}(:k[A

A-Za-z]+)?"} }?
```

xtent and name of the block.

lock.

```
 text },

{ text }} * }?
```

ed sequence, with attributes to describe the name and sequence.

```
d-sequence {
        attribute old { text },
         attribute new { text },
         attribute version { text }} * }?
```

8 Standardized Variants

The `standardized-variants` child of the `ucd` describes the standardized variant. It has one child element `standardized-variant` per variant. The attributes on that last element capture the variation sequence, the description of the desired appearance, and the shaping environment under which the appearance is different.

[standardized variants, 48] = `ucd.content &=`
```
xt },


tribute when { text }} * }?


 * }?
```

ces we have described so far:
6] *[repertoire: 7, 8, 9, 10, 11, 12]* *[properties: 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44]* *[blocks: 45]* *[named sequences:*

: 46] *[normalization corrections: 47]*

[standardized variants:

riants: 48]

gment of the UCD for a few representative characters (only some of the properties are
operties are represented):
 cp="001F" age="1.1" na="&lt;control&gt;" na1="UNIT SEPARATOR"
p="001F" age="1.1" na="&lt;control&gt;" na1="UNIT SEPARATOR"
"Cc" bc="S" lb="CM"/>
c" bc="S" lb="CM"/>
bc="WS" ea="Na" lb="SP"/>
="WS" ea="Na" lb="SP"/>
 bc="ON" ea="Na"/>
HESIS" na1="OPENING PARENTHESIS"
SIS" na1="OPENING PARENTHESIS"
idi_M="y" bmg="0029" ea="Na" lb="OP"/>
r cp="0041" age="1.1" na="LATIN CAPITAL LETTER A"
cp="0041" age="1.1" na="LATIN CAPITAL LETTER A"
Lu" slc="0061" ea="Na" sc="Latn"/>
"2.0" na="HANGUL SYLLABLE GA" gc="Lo"
.0" na="HANGUL SYLLABLE GA" gc="Lo"
1161" ea="W" lb="ID" sc="Hang"/>
3.1" na="CJK UNIFIED IDEOGRAPH-20094"
 lb="ID" sc="Hani" kIRG_GSource="KX"
GHanyuDaZidian="10036.060" kIRG_TSource="5-214E"
RG_TSource="5-214E"
5-214E"
214E"
ngXi="0082.090">
/>
" sc="Buhd">

="BUHID VOWEL SIGN I" gc="Mn"/>

EL SIGN I" gc="Mn"/>

cp="1820" age="3.0" na="MONGOLIAN LETTER A" sc="Mong"/>

ments

[Feedback] http://www.unicode.org/reporting.html
For reporting errors and requesting information online.

[ISO 19757] ISO/IEC 19757-2:2003 - Information technology - Document Schema Definition Language (DSDL) - Part 2: Regular-grammar-based validation - RELAX NG
Available at http://standards.iso.org/ittf/PubliclyAvailableStandards.

[ISO 19757 Amd1] ISO/IEC 19757-2:2003 - Information technology - Document Schema Definition Language (DSDL) - Part 2: Regular-grammar-based validation - RELAX NG - Amendment 1: Compact Syntax
Available at http://standards.iso.org/ittf/PubliclyAvailableStandards.

[Reports] Unicode Technical Reports

to Markus Scherer and Mark Davis for their help developing this XML representation.

http://www.unicode.org/reports/
For information on the status and development process for technical reports, and for a list of technical reports.

[Unicode]	The Unicode Consortium. *The Unicode Standard, Version 5.0* (Boston, MA, Addison-Wesley, 2007. ISBN 0-321-48091-0)
[Versions]	Versions of the Unicode Standard http://www.unicode.org/versions/ *For details on the precise contents of each version of the Unicode Standard, and how to cite them.*

Modifications

This section indicates the changes introduced by each revision.

Revision 1

- First version