Technical Reports

L2/08-066

Draft Unicode Standard Annex #42

UNICODE CHARACTER DATABASE IN XML

Version	5.1.0 Draft
Author	Eric Muller (emuller@adobe.com)
Date	2007-12-14
This Version	http://www.unicode.org/reports/tr42/tr42-2.html
Previous Version	http://www.unicode.org/reports/tr42/tr42-1.html
Latest Version	http://www.unicode.org/reports/tr42/
Schema	http://www.unicode.org/reports/tr42/tr42-2.rnc
Tracking Number	2

Summary

This annex 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*This is a **draft** document which may be updated, replaced, or superseded by other documents at any time.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 Standard Annex (UAX)** forms an integral part of the Unicode Standard, but is published online as a separate document. The Unicode Standard may require conformance to normative content in a Unicode Standard Annex, if so specified in the Conformance chapter of that version of the Unicode Standard. The version number of a UAX document corresponds to the version of the Unicode Standard of which it forms a part.*

Please submit corrigenda and other comments with the online reporting form [Feedback]. Related information that is useful in understanding this annex is found in Unicode Standard Annex #41, "Common References for Unicode Standard Annexes.". 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.

Directly accessing the data files that constitute the UCD is sometimes 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 (for example, 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 as well.

This annex 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 as 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 in using the UCD data to process 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 XML document with respect to the schema defined in this annex 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 support 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;", that is 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; for example, 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 in 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 [Versions]; 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, that is 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 noncharacters
- 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 makes answering questions like “what is the age of U+1749?” easier, because 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, that is “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 =
  element group {
    code-point-properties,
    code-point* }
```

4.4 Properties

Each property, except for the Block and Special_Case_Condition properties, is represented by an attribute.

The name of the attribute is the abbreviated name of the property as given in the file PropertyAliases.txt in version 5.0.0 of the UCD. For the 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).

For catalog and enumerated properties, the values are those listed in the file PropertyValueAliases.txt in version 5.0.0 of the UCD; if there is an abbreviated name, it is used, otherwise the long name is used.

4.4.1 Age property

The `age` attribute captures the version of Unicode in which a code point was assigned to an abstract character, or made a surrogate or non-character.

[age, 13] =

```
code-point-properties &= attribute age {
        "1.1" | "2.0" | "2.1" | "3.0" | "3.1" | "3.2"
      | "4.0" | "4.1" | "5.0" | "5.1" | "unassigned" }?
```

4.4.2 Name Properties

There are two name properties: the name given by the current version of the standard (`na`), and possibly the name this character had in version 1.0 of the standard (`na1`).

[name pattern, 14] =

```
  character-name = xsd:string { pattern="([A-Z0-9 #\-\(\)]*)|(<control>)" }
```

[name properties, 15] =

```
  code-point-properties &= attribute na { character-name }?
  code-point-properties &= attribute na1 { character-name }?
```

The majority of the characters in Unicode have a name which is of the form CJK UNIFIED IDEOGRAPH-<code point>. It also happens that character names cannot contain the character U+0023 # NUMBER SIGN, so we adopted the following convention: if a code point 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 value of the code point. For example:

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

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 &=
    attribute gc { "Lu" | "Ll" | "Lt" | "Lm" | "Lo"
                 | "Mn" | "Mc" | "Me"
                 | "Nd" | "Nl" | "No"
                 | "Pc" | "Pd" | "Ps" | "Pe" | "Pi" | "Pf" | "Po"
                 | "Sm" | "Sc" | "Sk" | "So"
                 | "Zs" | "Zl" | "Zp"
                 | "Cc" | "Cf" | "Cs" | "Co" | "Cn"
    }?
```

4.4.4 Combining Properties

The combining class is represented by the `ccc` attribute, which holds the decimal representation of the combining class.

Because the set of values that this property has taken accross the various versions of the UCD is rather large, our schema does not restrict the possible values to those actually used.

[ccc property, 17] =

```
  code-point-properties &=
    attribute ccc { xsd:integer { minInclusive="0" maxInclusive="255" }}?
```

4.4.5 Bidirectionality Properties

The bidirectional category is represented by the `bc` attribute.

[bc property, 18] =

```
  code-point-properties &=
    attribute bc { "AL" | "AN"
                 | "B " | "BN"
                 | "CS"
                 | "EN" | "ES"  | "ET"
                 | "L"  | "LRE" | "LRO"
                 | "NSM"
                 | "ON"
                 | "PDF"
                 | "R"  | "RLE" | "RLO"
                 | "S"
                 | "WS"
     }?
```

The mirrored property is represented by the `Bidi_M` attribute, which takes a boolean value.

[bidi_M property, 19] =

```
  code-point-properties &=
    attribute Bidi_M { boolean }?
```

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 image of the glyph for the current character.

[bmg property, 20] =

```
  code-point-properties &=
    attribute bmg    { "" | single-code-point }?
```

Note that we do not express the “Best Fit” element recorded 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 makes assumptions about the design of the fonts, and the best fit goes even farther.

The Bidi_Control property is represented by the `bidi_c` attribute.

[Bidi_C property, 21] =

```
  code-point-properties &=
    attribute Bidi_C { boolean }?
```

4.4.6 Decomposition Properties

The decomposition type and decomposition mapping properties are represented by the `dt` and `dm` attributes.

Most characters have a decomposition mapping to themselves. This is very similar to the situation we encountered with names, and we adopted a similar convention: if the value of a decomposition mapping is 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.

[decomposition properties, 22] =

```
  code-point-properties &=
    attribute dt { "can"  | "com" | "enc" | "fin"  | "font" | "fra"
                 | "init" | "iso" | "med" | "nar"  | "nb"   | "sml"
                 | "sqr"  | "sub" | "sup" | "vert" | "wide" | "none"}?
  code-point-properties &=
    attribute dm { "#" | zero-or-more-code-points }?
```

The properties Composition_Exclusion and Full_Composition_Exclusion are represented by the attributes `CE` and `Comp_Ex`:

[composition properties, 23] =

```
  code-point-properties &=
    attribute CE { boolean }?
  code-point-properties &=
    attribute Comp_Ex { boolean }?
```

The properties NFC_Quick_Check, NFD_Quick_Check, NFKC_Quick_Check, NFKD_Quick_Check, Expands_On_NFC, Expands_On_NFD, Expands_On_NFKC, Expands_On_NKFD, FC_NFKC_Closure have corresponding attribues.

[quick check properties, 24] =

```
  code-point-properties &=
    attribute NFC_QC { "Y" | "N" | "M" }?
  code-point-properties &=
    attribute NFD_QC { "Y" | "N" }?
  code-point-properties &=
    attribute NFKC_QC { "Y" | "N" | "M" }?
  code-point-properties &=
    attribute NFKD_QC { "Y" | "N" }?
  code-point-properties &=
    attribute XO_NFC { boolean }?
  code-point-properties &=
    attribute XO_NFD { boolean }?
  code-point-properties &=
    attribute XO_NFKC { boolean }?
  code-point-properties &=
    attribute XO_NFKD { boolean }?
  code-point-properties &=
    attribute FC_NFKC { zero-or-more-code-points }?
```

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 &=
    attribute nt { "None" | "De" | "Di" | "Nu" }?
  code-point-properties &=
    attribute nv { "" | xsd:string { pattern = "-?[0-9]+(/[0-9]+)?" }}?
```

4.4.8 Joining Properties

The joining class of a character is represented by the `jt` attribute.

If the joining class is neither “U”, “C” nor “T”, then the `jg` attribute is the joining group of the character.

[joining properties, 26] =

```
  code-point-properties &=
    attribute jt { "U" | "C" | "T" | "D" | "L" | "R" }?
  code-point-properties &=
    attribute jg { "Ain" | "Alaph" | "Alef" | "Alef_Maqsurah"
                 | "Beh" | "Beth" | "Burushaski_Yeh_Barree"
                 | "Dal" | "Dalath_Rish" | "E"
                 | "Fe" | "Feh" | "Final_Semkath"
                 | "Gaf" | "Gamal"
                 | "Hah" | "Hamza_On_Heh_Goal" | "He"
                 | "Heh" | "Heh_Goal" | "Heth"
                 | "Kaf" | "Kaph" | "Khaph" | "Knotted_Heh"
                 | "Lam" | "Lamadh" | "Meem" | "Mim"
                 | "No_Joining_Group" | "Noon" | "Nun"
                 | "Pe" | "Qaf" | "Qaph" | "Reh" | "Reversed_Pe"
                 | "Sad" | "Sadhe" | "Seen" | "Semkath" | "Shin"
                 | "Swash_Kaf" | "Syriac_Waw" | "Tah" | "Taw"
                 | "Teh_Marbuta" | "Teth" | "Waw" | "Yeh"
                 | "Yeh_Barree" | "Yeh_With_Tail" | "Yudh"
                 | "Yudh_He" | "Zain" | "Zhain" }?
```

The Join_Control property is representedy by the `Join_C` attribute.

[joining properties, 27] =

```
  code-point-properties &=
    attribute Join_C { boolean }?
```

4.4.9 Linebreak Properties

The linebreak property is represented by the `lb` attribute.

[linebreak property, 28] =

```
  code-point-properties &=
    attribute lb { "AI" | "AL" | "B2" | "BA" | "BB" | "BK" | "CB"
                 | "CL" | "CM" | "CR" | "EX" | "GL" | "H2" | "H3"
                 | "HY" | "ID" | "IN" | "IS" | "JL" | "JT" | "JV"
                 | "LF" | "NL" | "NS" | "NU" | "OP" | "PO" | "PR"
                 | "QU" | "SA" | "SG" | "SP" | "SY" | "WJ" | "XX"
                 | "ZW" }?
```

4.4.10 East Asian Width Propertie

The East Asian width property is represented by the `ea` attribute.

[ea property, 29] =

```
  code-point-properties &=
    attribute ea { "A" | "F" | "H" | "N" | "Na" | "W" }?
```

4.4.11 Case Properties

The Uppercase, Lowercase, Other_Uppercase and Other_Lowercase properties are represented by corresponding attributes.

[casing properties, 30] =

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

Most characters have a case mapping and case folding properties that simply map or fold to themselves. This is very similar to the situation we encountered with names, and we adopted a similar convention: if the value of a case mapping or case folding property 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.

The simple case mappings are recorded in the `suc`, `slc`, `stc` attributes.

[casing properties, 31] =

```
  code-point-properties &=
    attribute suc { "#" | single-code-point }?
  code-point-properties &=
    attribute slc { "#" | single-code-point }?
  code-point-properties &=
    attribute stc { "#" | single-code-point }?
```

The non-simple casing are recorded in the `uc`, `lc` and `tc` attributes.

[casing properties, 32] =

```
  code-point-properties &=
    attribute uc { "#" | one-or-more-code-points }?
  code-point-properties &=
    attribute lc { "#" | one-or-more-code-points }?
  code-point-properties &=
    attribute tc { "#" | one-or-more-code-points }?
```

The Simple_Case_Folding and Case_Folding properties are recorded in the `scf` and `cf` attributes respectively.

[casing properties, 33] =

```
  code-point-properties &=
    attribute scf { "#" | single-code-point }?
  code-point-properties &=
    attribute cf { "#" | one-or-more-code-points }?
```

Note that the UCD records more information about case folding than is expressed in the properties, specifically the entries in CaseFolding.txt with status T.

4.4.12 Script Properties

The script property is represented by the `sc` attribute.

[script property, 34] =

```
  code-point-properties &=
    attribute sc  { "Arab" | "Armn"
                  | "Bali" | "Beng" | "Bopo" | "Brai" | "Bugi" | "Buhd"
                  | "Cans" | "Cari" | "Cham" | "Cher" | "Copt" | "Cprt"
                  | "Cyrl"
                  | "Deva" | "Dsrt"
                  | "Ethi"
                  | "Geor" | "Glag" | "Goth" | "Grek" | "Gujr" | "Guru"
                  | "Hang" | "Hani" | "Hano" | "Hebr" | "Hira" | "Hrkt"
                  | "Ital"
                  | "Kali" | "Kana" | "Khar" | "Khmr" | "Knda"
                  | "Laoo" | "Latn" | "Lepc" | "Limb" | "Linb" | "Lyci"
                  | "Lydi"
                  | "Mlym" | "Mong" | "Mymr"
                  | "Nkoo"
                  | "Ogam" | "Olck" | "Orya" | "Osma"
                  | "Phag" | "Phnx"
                  | "Qaai"
                  | "Rjng" | "Runr"
                  | "Saur" | "Shaw" | "Sinh" | "Sund" | "Sylo" | "Syrc"
                  | "Tagb" | "Tale" | "Talu" | "Taml" | "Telu" | "Tfng"
```

```
                    | "Tglg" | "Thaa" | "Thai" | "Tibt"
                    | "Ugar"
                    | "Vai "
                    | "Xpeo" | "Xsux"
                    | "Yiii"
                    | "Zyyy" | "Zzzz" }?
```

4.4.13 ISO Comment Properties

The ISO 10646 comment field is represented by the `iso` attribute.

[isc property, 35] =

```
  code-point-properties &=
    attribute isc { text }?
```

4.4.14 Hangul Properties

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

[hst property, 36] =

```
  code-point-properties &=
    attribute hst { "L" | "LV" | "LVT" | "T" | "V" | "NA" }?
```

The property Jamo_Short_Name is represented by the `JSN` attribute:

[jamo property, 37] =

```
  code-point-properties &=
     attribute JSN { xsd:string { pattern="[A-Z]{0,3}" }}?
```

4.4.15 Identifier and Pattern and programming language properties

The properties ID_Start, Other_ID_Start, XID_Start , ID_Continue, Other_ID_Continue, and XID_Continue are represented by corresponding attributes:

[identifier properties, 38] =

```
  code-point-properties &=
    attribute IDS { boolean }?
  code-point-properties &=
    attribute OIDS { boolean }?
  code-point-properties &=
    attribute XIDS { boolean }?
  code-point-properties &=
    attribute IDC { boolean }?
  code-point-properties &=
    attribute OIDC { boolean }?
  code-point-properties &=
    attribute XIDC { boolean }?
```

The properties Pattern_Syntax and Pattern_White_Space are represented by corresponding attributes:

[pattern properties, 39] =

```
  code-point-properties &=
    attribute Pat_Syn { boolean }?
  code-point-properties &=
    attribute Pat_WS { boolean }?
```

4.4.16 Properties related to function and graphic characteristics

The properties Dash, Hyphen, Quotation_Mark, Terminal_Punctuation, STerm, Diacritic, Extender, Soft_Dotted, Alphabetic, Other_Alphabetic, Math, Other_Math, Hex_Digit, ASCII_Hex_Digit, Default_Ignorable_Code_Point, Other_Default_Ignorable_Code_Point, Logical_Order_Exception and White_Space describe the function or graphic characteristic of a character, and have each a corresponding attribute.

[properties related to function and graphic characteristics, 40] =

```
  code-point-properties &=
    attribute Dash { boolean }?
  code-point-properties &=
    attribute Hyphen { boolean }?
  code-point-properties &=
    attribute QMark { boolean }?
  code-point-properties &=
    attribute Term { boolean }?
  code-point-properties &=
    attribute STerm { boolean }?
  code-point-properties &=
    attribute Dia { boolean }?
  code-point-properties &=
    attribute Ext { boolean }?
  code-point-properties &=
    attribute SD { boolean }?
  code-point-properties &=
    attribute Alpha { boolean }?
  code-point-properties &=
    attribute OAlpha { boolean }?
  code-point-properties &=
    attribute Math { boolean }?
  code-point-properties &=
    attribute OMath { boolean }?
  code-point-properties &=
    attribute Hex { boolean }?
  code-point-properties &=
    attribute AHex { boolean }?
  code-point-properties &=
    attribute DI { boolean }?
  code-point-properties &=
    attribute ODI { boolean }?
  code-point-properties &=
    attribute LOE { boolean }?
  code-point-properties &=
    attribute WSpace { boolean }?
```

4.4.17 Properties related to boundaries

The properties Grapheme_Base, Grapheme_Extend, Other_Grapheme_Extend, Grapheme_Link, Grapheme_Cluster_Break, Word_Break and Sentence_Break each have a corresponding attribute:

[properties related to boundaries, 41] =

```
  code-point-properties &=
    attribute Gr_Base { boolean }?
  code-point-properties &=
    attribute Gr_Ext { boolean }?
  code-point-properties &=
    attribute OGr_Ext { boolean }?
  code-point-properties &=
    attribute Gr_Link { boolean }?
  code-point-properties &=
    attribute GCB { "CN" | "CR" | "EX" | "L"  | "LF" |  "LV" | "LVT"
                  | "SM" | "T"  | "V"  | "XX" }?
  code-point-properties &=
    attribute WB { "CR" | "EX" | "Extend" | "FO" | "KA" | "LE"
                 | "LF" | "MB" | "ML" | "MN" |  "NU" | "XX" }?
  code-point-properties &=
    attribute SB { "AT" | "CL" | "CR" | "EX" | "FO" | "LE" | "LF" | "LO"
                 | "NU" | "SC" | "SE" | "SP" | "ST" | "UP" | "XX" }?
```

4.4.18 Properties related to ideographs

The properties Ideographic, Unified_Ideograph, IDS_Binary_Operator, IDS_Trinary_Operator and Radical have corresponding attributes:

[properties related to ideographs, 42] =

```
  code-point-properties &=
    attribute Ideo { boolean }?
  code-point-properties &=
    attribute UIdeo { boolean }?
  code-point-properties &=
    attribute IDSB { boolean }?
  code-point-properties &=
    attribute IDST { boolean }?
  code-point-properties &=
    attribute Radical { boolean }?
```

4.4.19 Miscellaneous properties

The properties Deprecated, Variation_Selector and Noncharacter_Code_Point have corresponding attributes:

[miscellaneous properties, 43] =

```
  code-point-properties &=
    attribute Dep { boolean }?
  code-point-properties &=
    attribute VS { boolean }?
  code-point-properties &=
    attribute NChar { boolean }?
```

4.4.20 Unihan properties

The Unihan properties (from Unihan.txt) are represented as attributes.

[Unihan properties, 44] =

```
  code-point-properties &= attribute kAccountingNumeric
     { xsd:string {pattern="[0-9]+"} }?
  code-point-properties &= attribute kAlternateHanYu
     { text }?  #old
  code-point-properties &= attribute kAlternateJEF
     { text }?  #old
  code-point-properties &= attribute kAlternateKangXi
     { text }?
  code-point-properties &= attribute kAlternateMorohashi
     { text }?
  code-point-properties &= attribute kBigFive
     { xsd:string {pattern="[0-9A-F]{4}"} }?
  code-point-properties &= attribute kCCCII
     { xsd:string {pattern="[0-9A-F]{6}"} }?
  code-point-properties &= attribute kCNS1986
     { xsd:string {pattern="[12E]-[0-9A-F]{4}"} }?
  code-point-properties &= attribute kCNS1992
     { xsd:string {pattern="[123]-[0-9A-F]{4}"} }?
  code-point-properties &= attribute kCangjie
     { xsd:string {pattern="[A-Z]+"} }?
  code-point-properties &= attribute kCantonese
     { list { xsd:string {pattern="[a-z]+[1-6]"} +}}?
  code-point-properties &= attribute kCheungBauer
     { text }?
  code-point-properties &= attribute kCheungBauerIndex
     { list { xsd:string {pattern="[0-9]{3}\.[0-9]{2}"} +}}?
  code-point-properties &= attribute kCihaiT
     { list { xsd:string {pattern="[1-9][0-9]{0,3}\.[0-9]{3}"} +}}?
  code-point-properties &= attribute kCompatibilityVariant
     { "" | xsd:string {pattern="U\+2?[0-9A-F]{4}"} }?
  code-point-properties &= attribute kCowles
```

```
   { list { xsd:string {pattern="[0-9]{1,4}(\.[0-9]{1,2})?"} +}}?
code-point-properties &= attribute kDaeJaweon
   { xsd:string {pattern="[0-9]{4}\.[0-9]{2}[0158]"} }?
code-point-properties &= attribute kDefinition
   { text }?
code-point-properties &= attribute kEACC
   { xsd:string {pattern="[0-9A-F]{6}"} }?
code-point-properties &= attribute kFenn
   { list { xsd:string {pattern="[0-9]+a?[A-KP*]"} +}}?
code-point-properties &= attribute kFennIndex
   { list { xsd:string {pattern="[1-9][0-9]{0,2}\.[01][0-9]"} +}}?
code-point-properties &= attribute kFourCornerCode
   { list { xsd:string {pattern="[0-9]{4}(\.[0-9])?"} +}}?
code-point-properties &= attribute kFrequency
   { xsd:string {pattern="[1-5]"} }?
code-point-properties &= attribute kGB0
   { xsd:string {pattern="[0-9A-F]{4}"} }?
code-point-properties &= attribute kGB1
   { xsd:string {pattern="[0-9A-F]{4}"} }?
code-point-properties &= attribute kGB3
   { xsd:string {pattern="[0-9A-F]{4}"} }?
code-point-properties &= attribute kGB5
   { xsd:string {pattern="[0-9A-F]{4}"} }?
code-point-properties &= attribute kGB7
   { xsd:string {pattern="[0-9A-F]{4}"} }?
code-point-properties &= attribute kGB8
   { xsd:string {pattern="[0-9]{4}"} }?
code-point-properties &= attribute kGradeLevel
   { xsd:string {pattern="[1-6]"} }?
code-point-properties &= attribute kGSR
   { list { xsd:string {pattern="[0-9]{4}[a-vx-z]'*"} +}}?
code-point-properties &= attribute kHangul
   { text }?
code-point-properties &= attribute kHanYu
   { list { xsd:string {pattern="[1-8][0-9]{4}\.[0-9]{2}[0-3]"} +}}?
code-point-properties &= attribute kHanyuPinlu
   { list { xsd:string {pattern="[a-zü]+[1-5]\([0-9]+\)"} +}}?
code-point-properties &= attribute kHDZRadBreak
   { xsd:string {pattern="[一-龠]\[U\+2?[0-9A-F]{4}\]:[1-8][0-9]{4}\.[0-9]{2}[012]"} }?
code-point-properties &= attribute kHKGlyph
   { list { xsd:string {pattern="[0-9]{4}"} +}}?
code-point-properties &= attribute kHKSCS
   { xsd:string {pattern="[0-9A-F]{4}"} }?
code-point-properties &= attribute kIBMJapan
   { xsd:string {pattern="F[ABC][0-9A-F]{2}"} }?
code-point-properties &= attribute kIICore
   { xsd:string {pattern="[1-9]\.[1-9]"} }?
code-point-properties &= attribute kIRGDaeJaweon
   { xsd:string {pattern="([0-9]{4}\.[0-9]{2}[01])|(0000\.555)"} }?
code-point-properties &= attribute kIRGDaiKanwaZiten
   { xsd:string {pattern="[0-9]{5}'?"} }?
code-point-properties &= attribute kIRGHanyuDaZidian
   { xsd:string {pattern="[1-8][0-9]{4}\.[0-3][0-9][01]"} }?
code-point-properties &= attribute kIRGKangXi
   { xsd:string {pattern="[01][0-9]{3}\.[0-7][0-9][01]"} }?
code-point-properties &= attribute kIRG_GSource
   { "" | xsd:string {pattern="(4K|BK|CH|CY|FZ(_BK)?|HC|HZ|KX|[0135789ES]-[0-9A-F]{4})"} }
code-point-properties &= attribute kIRG_HSource
   { "" | xsd:string {pattern="[0-9A-F]{4}"} }?
code-point-properties &= attribute kIRG_JSource
   { "" | xsd:string {pattern="([0134A]|3A)-[0-9A-F]{4}"} }?
code-point-properties &= attribute kIRG_KPSource
   { "" | xsd:string {pattern="KP[01]-[0-9A-F]{4}"} }?
code-point-properties &= attribute kIRG_KSource
   { "" | xsd:string {pattern="[01234]-[0-9A-F]{4}"} }?
code-point-properties &= attribute kIRG_TSource
   { "" | xsd:string {pattern="[1-7F]-[0-9A-F]{4}"} }?
code-point-properties &= attribute kIRG_USource
   { "" | xsd:string {pattern="(U\+2?[0-9A-F]{4})|(UTC[0-9]{5})"} }?
code-point-properties &= attribute kIRG_VSource
   { "" | xsd:string {pattern="[0123]-[0-9A-F]{4}"} }?
code-point-properties &= attribute kJHJ
   { text }?
```

```
code-point-properties &= attribute kJIS0213
   { xsd:string {pattern="[12],[0-9]{2},[0-9]{1,2}"} }?
code-point-properties &= attribute kJapaneseKun
   { list { xsd:string {pattern="[A-Z]+"}+ } }?
code-point-properties &= attribute kJapaneseOn
   { list { xsd:string {pattern="[A-Z]+"}+ } }?
code-point-properties &= attribute kJis0
   { xsd:string {pattern="[0-9]{4}"} }?
code-point-properties &= attribute kJis1
   { xsd:string {pattern="[0-9]{4}"} }?
code-point-properties &= attribute kKPS0
   { xsd:string {pattern="[0-9A-F]{4}"} }?
code-point-properties &= attribute kKPS1
   { xsd:string {pattern="[0-9A-F]{4}"} }?
code-point-properties &= attribute kKSC0
   { xsd:string {pattern="[0-9]{4}"} }?
code-point-properties &= attribute kKSC1
   { xsd:string {pattern="[0-9]{4}"} }?
code-point-properties &= attribute kKangXi
   { xsd:string {pattern="[0-9]{4}\.[0-9]{2}[01]"} }?
code-point-properties &= attribute kKarlgren
   { xsd:string {pattern="[1-9][0-9]{0,3}[A*]?"} }?
code-point-properties &= attribute kKorean
   { list { xsd:string {pattern="[A-Z]+"} +}}?
code-point-properties &= attribute kLau
   { list { xsd:string {pattern="[1-9][0-9]{0,3}"} +}}?
code-point-properties &= attribute kMainlandTelegraph
   { xsd:string {pattern="[0-9]{4}"} }?
code-point-properties &= attribute kMandarin
   { list { xsd:string {pattern="[A-ZÜ]+[1-5]"} +}}?
code-point-properties &= attribute kMatthews
   { xsd:string {pattern="[0-9]{1,4}(a|\.5)?"} }?
code-point-properties &= attribute kMeyerWempe
   { list { xsd:string {pattern="[1-9][0-9]{0,3}[a-t*]?"} +}}?
code-point-properties &= attribute kMorohashi
   { xsd:string {pattern="[0-9]{5}'?"} }?
code-point-properties &= attribute kNelson
   { list { xsd:string {pattern="[0-9]{4}"} +}}?
code-point-properties &= attribute kOtherNumeric
   { list { xsd:string {pattern="[0-9]+"} +}}?
code-point-properties &= attribute kPhonetic
   { list { xsd:string {pattern="[1-9][0-9]{0,3}[A-D]?\*?"} +}}?
code-point-properties &= attribute kPrimaryNumeric
   { xsd:string {pattern="[0-9]+"} }?
code-point-properties &= attribute kPseudoGB1
   { xsd:string {pattern="[0-9]{4}"} }?
code-point-properties &= attribute kRSAdobe_Japan1_6
   { list { xsd:string {pattern="[CV]\+[0-9]{1,5}\+[1-9][0-9]{0,2}\.[1-9][0-9]?\.[0-9]{1,2
code-point-properties &= attribute kRSJapanese
   { xsd:string {pattern="[0-9]{1,3}\.[0-9]{1,2}"} }?
code-point-properties &= attribute kRSKanWa
   { xsd:string {pattern="[0-9]{1,3}\.[0-9]{1,2}"} }?
code-point-properties &= attribute kRSKangXi
   { xsd:string {pattern="[0-9]{1,3}\.[0-9]{1,2}"} }?
code-point-properties &= attribute kRSKorean
   { xsd:string {pattern="[0-9]{1,3}\.[0-9]{1,2}"} }?
code-point-properties &= attribute kRSMerged
   { text }?
code-point-properties &= attribute kRSUnicode
   { list { xsd:string {pattern="[0-9]{1,3}'?\.[0-9]{1,2}"} +}}?
code-point-properties &= attribute kSBGY
   { list { xsd:string {pattern="[0-9]{3}\.[0-9]{2}"} +}}?
code-point-properties &= attribute kSemanticVariant
   { list { xsd:string {pattern="U\+2?[0-9A-F]{4}(<k[A-Za-z:]+(,k[A-Za-z]+)*)?"} +}}?
code-point-properties &= attribute kSimplifiedVariant
   { list { xsd:string {pattern="U\+2?[0-9A-F]{4}"} +}}?
code-point-properties &= attribute kSpecializedSemanticVariant
   { list { xsd:string {pattern="U\+2?[0-9A-F]{4}(<k[A-Za-z]+(,k[A-Za-z]+)*)?"} +}}?
code-point-properties &= attribute kTaiwanTelegraph
   { xsd:string {pattern="[0-9]{4}"} }?
code-point-properties &= attribute kTang
   { list { xsd:string {pattern="\*?[A-Za-z\(\)æɑəɛ̌]+"} +}}?
code-point-properties &= attribute kTotalStrokes
```

```
     { xsd:string {pattern="[1-9][0-9]{0,2}"} }?
  code-point-properties &= attribute kTraditionalVariant
     { list { xsd:string {pattern="U\+2?[0-9A-F]{4}"} +}}?
  code-point-properties &= attribute kVietnamese
     { list { xsd:string {pattern="[A-Za-zà-ưạ-ỹ]+"} +}}?
  code-point-properties &= attribute kXHC1983
     { list { xsd:string {pattern="[0-9,.*]+:[a-zǜ̂]+"} +}} ?
  code-point-properties &= attribute kWubi
     { text }?
  code-point-properties &= attribute kXerox
     { xsd:string {pattern="[0-9]{3}:[0-9]{3}"} }?
  code-point-properties &= attribute kZVariant
     { xsd:string {pattern="U\+2?[0-9A-F]{4}(:k[A-Za-z]+)?"} }?
```

5 Blocks

The `blocks` child of the `ucd` describes the blocks. It has one child `block` element per block, with attributes to describe the extent and name of the block.

[blocks, 45] =

```
  ucd.content &=
    element blocks {
      element block {
        attribute first-cp { text },
        attribute last-cp { text },
        attribute name { text }} * }?
```

6 Named Sequences

The `named-sequences` child of the `ucd` describes the named sequences. It has one child `named-sequence` element per named sequence, with attributes to describe the name and sequence.

Similarly, the `provisional-named-sequences` child of the `ucd` describes the provisional named sequences.

[named sequences, 46] =

```
  ucd.content &=
    element named-sequences {
      element named-sequence {
        attribute cps { text },
        attribute name { text }} * }?
  ucd.content &=
    element provisional-named-sequences {
      element named-sequence {
        attribute cps { text },
        attribute name { text }} * }?
```

7 Normalization Corrections

The `normalization-corrections` child of the `ucd` describes the normalization corrections. It has one child `normalization-correction` element per correction, with attributes to describe the code point affected, its old normalization, its new normalization and the version of Unicode in which the correction was made.

[normalization corrections, 47] =

```
  ucd.content &=
    element normalization-corrections {
      element normalization-correction {
        attribute cp { text },
        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 &=
    element standardized-variants {
      element standardized-variant {
        attribute cps { text },
        attribute desc { text },
        attribute when { text }} * }?
```

9 The full schema

Our schema is just the accumulation of the pieces we have described so far:

[UCD RelaxNG schema, 49] =

[namespace declaration: 1]
[datatypes: 2, 3, 14]
[schema start: 4]
[boolean type: 5]
[description: 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: 48]

An expanded version is linked from the top of this document.

10 Examples

Here is a fragment of the UCD for a few representative characters (only some of the properties are represented):

```
<char cp="001F" age="1.1" na="&lt;control&gt;" na1="UNIT SEPARATOR"
    gc="Cc" bc="S" lb="CM"/>

<char cp="0020" age="1.1" na="SPACE" gc="Zs" bc="WS" ea="Na" lb="SP"/>

<char cp="0026" age="1.1" na="AMPERSAND" gc="Po" bc="ON" ea="Na"/>

<char cp="0028" age="1.1" na="LEFT PARENTHESIS" na1="OPENING PARENTHESIS"
    gc="Ps" bc="ON" Bidi_M="y" bmg="0029" ea="Na" lb="OP"/>

<char cp="0041" age="1.1" na="LATIN CAPITAL LETTER A"
    gc="Lu" slc="0061" ea="Na" sc="Latn"/>

<char cp="AC00" age="2.0" na="HANGUL SYLLABLE GA" gc="Lo"
```

```
      dt="can" dm="1100 1161" ea="W" lb="ID" sc="Hang"/>

  <char cp="20094" age="3.1" na="CJK UNIFIED IDEOGRAPH-20094"
      gc="Lo" ea="W" lb="ID" sc="Hani" kIRG_GSource="KX"
      kIRGHanyuDaZidian="10036.060" kIRG_TSource="5-214E"
      kRSUnicode="4.3" kIRGKangXi="0082.090">
    <kPhonetic v="0139"/>
  </char>

 <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>
```

Appendix A. Acknowledgments

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

Modifications

This section indicates the changes introduced by each revision.

Revision 2

- Promoted to Draft UAX.
- Changed the title from "An XML representation of the UCD"
- Value `5.1` added to the `age` attribute (for Unicode 5.1).
- Value `SM` added to the `gcb` attribute (for Unicode 5.1).
- Values `CR`, `Extend`, `LF`, `MB` added to the `WB` attribute (for Unicode 5.1).
- Values `CR`, `EX`, `LF`, `SC` added to the `SB` attribute (for Unicode 5.1).
- Value `Burushaski_Yeh_Barree` added to the `jg` attribute (for Unicode 5.1).
- Value `Alef_Maqsurah` added to the `jg` attribute (for Unicode 2.x).
- Values `Cari`, `Cham`, `Kali`, `Lepc`, `Lyci`, `Lydi`, `Olck`, `Rjng`, `Saur`, `Sund` and `Vai` added to the `sc` attribute (for Unicode 5.0).
- `jamo` attribute renamed to `JSN`
- `sfc` attribute renamed to `scf`
- Attribute `kXHC1983` added (for Unicode 5.1.0).
- Pattern for attribute `kIRG_USource` extended (for Unicode 5.1.0).
- Element `provisional-named-sequences` added (for Unicode 5.0)

Revision 1

- First version