Internet-Draft
Intended Category: Informational
Expires in six months

Editor: Kurt D. Zeilenga
OpenLDAP Foundation
3 March 2003

Internationalized String Matching Rules for X.500
<draft-zeilenga-ldapbis-strmatch-02.txt>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

This document is intended to be submitted to the ITU for publication as an amendment to X.520 and published as an Informational RFC. Distribution of this memo is unlimited. Technical discussion of this document will take place on the IETF LDAP Revision Working Group mailing list <ietf-ldapbis@openldap.org>. Please send editorial comments directly to the author <Kurt@OpenLDAP.org>.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts. Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as ``work in progress.''

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>. The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

Abstract

The existing X.500 Directory Service technical specifications do not precisely define how string matching is to be performed. This has lead to a number of interoperability problems. This document provides string preparation profiles for standard syntaxes and matching rules defined in X.520.

This document is intended to be submitted to the ITU-T for publication as an amendment to X.520 and published as an Informational RFC.

Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119].

Character names in this document use the notation for code points and names from the Unicode Standard [UNICODE] and ISO/IEC 10646-1 [ISO10646]. For example, the letter "a" may be represented as either <U+0061> or <LATIN SMALL LETTER A>. In the lists of mappings and the prohibited characters, the "U+" is left off to make the lists easier to read. The comments for character ranges are shown in square brackets (such as "[CONTROL CHARACTERS]") and do not come from the standards.

Note: a glossary of terms used in Unicode and ISO/IEC 10646 can be found in [GLOSSARY]. Information on the ISO/IEC 10646/Unicode character encoding model can be found in [UTR17].

1. Introduction

1.1. Background

An X.500 matching rule [X.501] defines an algorithm for determining whether a presented value matches an attribute value in accordance with the criteria defined for the rule. The proposition may be evaluated to True, False, or Undefined.

```
True      - the attribute contains a matching value,

False     - the attribute contains no matching value,

Undefined - it cannot be determined whether the attribute contains
            a matching value or not.
```

For instance, the caseIgnoreMatch matching rule may be used to compare whether the commonName attribute contains a particular value without regard for case and insignificant spaces.

1.2. X.500 String Matching Rules

"X.520: Selected attribute types" [X.520] provides (amongst other things) value syntaxes and matching rules for comparing values commonly used in the Directory [X.500]. These specifications are inadequate for strings composed of characters from the Universal Character Set (UCS) [ISO10646], a superset of Unicode [UNICODE].

The CaseIgnoreMatch matching rule, for example, is simply defined as being a case insensitive comparison where insignificant spaces are ignored. For printableString, there is only one space character and case mapping is bijective, hence this definition is sufficient. However, for UCS-based string types such as universalString, this is not sufficient. For example, a case insensitive matching implementation which folded lower case characters to upper case would yield different different results than an implementation which used upper case to lower case folding. Or one implementation may view space as referring to only SPACE (U+0020), a second implementation may view any character with the space separator (Zs) property as a space, and another implementation may view any character with the whitespace (WS) category as a space.

The lack of precise specification for string matching has led to significant interoperability problems. When used in certificate chain validation, security vulnerabilities can arise. To address these problems, this document updates X.520 [X.520] with a detailed specification of string syntax and matching rule requirements.

1.3. Relationship to "stringprep"

The matching rule algorithms described in this document are based upon the "stringprep" approach [RFC3454]. In "stringprep", presented and stored values are first prepared for comparison and so that a character-by-character comparison yields the "correct" result.

The algorithm used here is a refinement of the "stringprep" [RFC3454] approach. The algorithm involves two additional preparation steps.

a) prior to applying the Unicode string preparation steps outlined in "stringprep", the string is transcoded to Unicode;

b) after applying the Unicode string preparation steps outlined in "stringprep", characters insignificant to the matching rules are removed.

Hence, preparation of strings for X.500 matching involves the following steps:

1) Transcode

2) Map
3) Normalize
4) Prohibit
5) Check Bidi (Bidirectional)
6) Insignificant Character Removal

These steps are described in Section 3. Section 2 details design considerations.

1.4. Relationship to X.500

This document updates X.520 [X.520] with additional normative and informative information. Sections 3, 4, and 5 are normative parts of this update. Other sections are informative.

Section 3 provides a specification for X.500 string preparation. It is intended to be added as a new section in X.520.

Section 4 replaces section 6.1 of X.520 [X.520]. It updates select string matching rules.

Section 5 replaces portions of section 6.2 of X.520 [X.520]. It updates select syntax-based matching rules.

2. Design Considerations

The X.500 string matching rule specification provided in Section 3 is designed to leverage the "stringprep" framework [RFC3454] for comparing of strings. As noted above, transcoding and space removal steps have been added.

This section describes the rationale for these and other design decisions.

2.1. Transcode

In the past, transcoding only occurred when all of the input strings were not encoded in the same character set. If all were encoded in the same character set, no transcoding was to be performed. Otherwise, all of the strings would be transcoded to one of character sets used.

As mappings between character sets, such as T.61 and UCS, are not bijective, this specification requires transliteration of all strings to a common character encoding set. UCS was the logical choice as all

other character sets (used in X.500) can be transcoded to it without information loss. None of the other character sets (used in X.500) offer this property.

2.2. Map

Code points which have no semantic meaning in normal text are mapped to nothing. Code points which are semantically equivalent in normal text are mapped to a single code point.

"Normal text", in this context, is viewed as text commonly held in attributes of Directory String syntax, such as identifiers, common names, and short descriptive text.

2.3. Normalize

Normalization is performed to ensure that comparison is always done between canonical-equivalent strings. As directory strings are often used as identifiers, we selected Form KC (compatibility composed) as it allows a greater number of strings to be treated as equivalent.

Unfortunately, this choice is not best for all applications. Additional matching rules which use different string preparation algorithms may be introduced in the future to better support these applications. In particular, matching rules which use Form C (composed) normalization instead of Form KC would also be generally useful. It may be desirable to add additional matching rules to X.500 which use Form C normalization.

2.4. Prohibit

TBD

2.5. Check bidi

TBD

2.6. Insignificant Character Removal

This step is used to remove insignificant characters from the string. Unlike the map step, which supports mapping of characters to nothing, this step allows removal of characters based upon their location in the string, surrounding characters in the string, and other factors.

3. String Preparation

The following six-step process SHALL be applied to each presented and attribute value in preparation for string match rule evaluation.

1) Transcode
2) Map
3) Normalize
4) Prohibit
5) Check bidi
6) Insignificant Character Removal

Failure in any step is be cause the assertion to be Undefined.

The character repertoire of this process is Unicode 3.2 [UNICODE].

3.1. Transcode

Each non-Unicode string value is transcoded to Unicode.

TeletexString values are transcoded to Unicode as described in [T61-UCS].

PrintableString value are transcoded directly to Unicode.

UniversalString, UTF8String, and bmpString values need not be transcoded as they are Unicode-based strings (in the case of bmpString, restricted to a subset of Unicode).

If the implementation is unable or unwilling to perform the transcoding as described above, or the transcoding fails, this step fails and the assertion is evaluated to Undefined.

The transcoded string is the output string.

3.2. Map

SOFT HYPHEN (U+00AD) and MONGOLIAN TODO SOFT HYPHEN (U+1806) code points are mapped to nothing. COMBINING GRAPHEME JOINER (U+034F) and VARIATION SELECTORs (U+180B-180D,FF00-FE0F) code points are also mapped to nothing. The OBJECT REPLACEMENT CHARACTER (U+FFFC) is mapped to nothing.

CHARACTER TABULATION (U+0009), LINE FEED (LF) (U+000A), LINE TABULATION (U+000B), FORM FEED (FF) (U+000C), CARRIAGE RETURN (CR) (U+000D), and NEXT LINE (NEL) (U+0085) are mapped to SPACE (U+0020).

All other control code points (e.g., Cc) or code points with a control function (e.g., Cf) are mapped to nothing.

ZERO WIDTH SPACE (U+200B) is mapped to nothing. All other code points with Separator (space, line, or paragraph) property (e.g, Zs, Zl, or Zp) are mapped to SPACE (U+0020).

For case ignore, numeric, and stored prefix string matching rules, characters are case folded per B.2 of [RFC3454].

3.3. Normalize

The input string is be normalized to Unicode Form KC (compatibility composed) as described in [UAX15].

3.4. Prohibit

All Unassigned, Private Use, and non-character code points are prohibited. Surrogate codes (U+D800-DFFFF) are prohibited.

The REPLACEMENT CHARACTER (U+FFFD) code point is prohibited.

The first code point of a string is probibited from being a combining character.

Empty strings are prohibited.

The step fails and the assertion is evaluated to Undefined if the input string contains any prohibited code point. The output string is the input string.

3.5. Check bidi

There are no bidirectional restrictions. The output string is the input string.

3.6. Insignificant Character Removal

In this step, characters insignificant to the matching rule are to be removed. The characters to be removed differ from matching rule to matching rule.

Section 3.6.1 applies to case ignore and exact string matching.
Section 3.6.2 applies to numericString matching.

Section 3.6.3 applies to telephoneNumber matching

3.6.1. Insignificant Space Removal

For the purposes of this section, a space is defined to be the SPACE (U+0020) code point followed by no combining marks.

> NOTE - The previous steps ensure that the string cannot contain any code points in the separator class, other than SPACE (U+0020).

The following spaces are regarded as not significant and are to be removed:

- leading spaces (i.e. those preceding the first character that is not a space);
- trailing spaces (i.e. those following the last character that is not a space);
- multiple consecutive spaces (these are taken as equivalent to a single space character).

(A string consisting entirely of spaces is equivalent to a string containing exactly one space.)

For example, removal of spaces from the Form KC string:
"<SPACE><SPACE>foo<SPACE><SPACE>bar<SPACE><SPACE>" would result in the output string:
"<SPACE>foo<SPACE>bar<SPACE>".

and the Form KC string:
"<SPACE><SPACE><SPACE>" would result in the output string:
"<SPACE>".

3.6.2. NumericString Insignificant Character Removal

For the purposes of this section, a space is defined to be the SPACE (U+0020) code point followed by no combining marks.

All spaces are regarded as not significant and are to be removed.

For example, removal of spaces from the Form KC string:
"<SPACE><SPACE>123<SPACE><SPACE>456<SPACE><SPACE>" would result in the output string:
"123456".

and the Form KC string:
"<SPACE><SPACE><SPACE>" would result in an empty output string.

3.6.3. TelephoneNumber Insignificant Character Removal

For the purposes of this section, a hyphen is defined to be HYPHEN-MINUS (U+002D), ARMENIAN HYPHEN (U+058A), HYPHEN (U+2010), NON-BREAKING HYPHEN (U+2011), MINUS SIGN (U+2212), SMALL HYPHEN-MINUS (U+FE63), or FULLWIDTH HYPHEN-MINUS (U+FF0D) code point followed by no combining marks and a space is defined to be the SPACE (U+0020) code point followed by no combining marks.

All hyphens and spaces are regarded as not significant and are to be removed.

4. String Matching Rules

In the matching rules specified in this section, all presented and stored string values are be prepared for matching as described in Section 3. String preparation produces strings suitable for character-by-character matching.

4.1. Case Exact / Ignore Match

The Case Exact Match rule compares for equality a presented string with an attribute value of type DirectoryString or one of the data types appearing in the choice type DirectoryString, e.g. UTF8String, without regards to insignificant spaces (3.4.1).

```
caseExactMatch MATCHING-RULE ::= {
    SYNTAX DirectoryString {ub-match}
    ID id-mr-caseExactMatch }
```

The Case Ignore Match rule compares for equality a presented string with an attribute value of type DirectoryString or one of the data types appearing in the choice type DirectoryString, e.g. UTF8String, without regard to the case (upper or lower) of the strings (e.g. "Dundee" and "DUNDEE" match) and insignificant spaces (3.4.1). The rule is identical to the caseExactMatch rule except upper case characters are folded to lower case during string preparation as discussed in 3.2.

```
caseIgnoreMatch MATCHING-RULE ::= {
    SYNTAX DirectoryString {ub-match}
    ID id-mr-caseIgnoreMatch }
```

Both rules return TRUE if the prepared strings are the same length and corresponding characters are identical.

4.2. Case Exact / Ignore Ordering Match

The Case Exact Ordering Match rule compares the collation order of a presented string with an attribute value of type DirectoryString or one of the data types appearing in the choice type DirectoryString, e.g. UTF8String, without regard to insignificant spaces (3.4.1).

```
caseExactOrderingMatch MATCHING-RULE ::= {
    SYNTAX DirectoryString {ub-match}
    ID id-mr-caseExactOrderingMatch }
```

The Case Ignore Ordering Match rule compares the collation order of a presented string an attribute value of type DirectoryString or one of the data types appearing in the choice type DirectoryString, e.g. UTF8String, without regard to the case (upper or lower) of the strings and insignificant spaces (3.4.1). The rule is identical to the caseExactOrderingMatch rule except upper case characters are folded to lower case during string preparation as discussed in 3.2.

```
caseIgnoreOrderingMatch MATCHING-RULE ::= {
    SYNTAX DirectoryString {ub-match}
    ID id-mr-caseIgnoreOrderingMatch }
```

Both rules return TRUE if the attribute value is "less" or appears earlier than the presented value, when the prepared strings are compared using Unicode code point collation order.

4.3. Case Exact / Ignore Substrings Match

The Case Exact Substrings Match rule determines whether a presented value is a substring of an attribute value of type DirectoryString or one of the data types appearing in the choice type DirectoryString, e.g. UTF8String, without regard to insignficant spaces (3.4.1).

```
caseExactSubstringsMatch MATCHING-RULE ::= {
    SYNTAX SubstringAssertion
    ID id-mr-caseExactSubstringsMatch }
SubstringAssertion ::= SEQUENCE OF CHOICE {
    initial [0] DirectoryString {ub-match},
    any [1] DirectoryString {ub-match},
    final [2] DirectoryString {ub-match},
control Attribute }
-- Used to specify interpretation of the following items
-- at most one initial and one final component
```

The Case Ignore Substrings Match rule determines whether a presented value is a substring of an attribute value of type DirectoryString or

one of the data types appearing in the choice type DirectoryString, e.g. UTF8String, without regard to the case (upper or lower) of the strings and insignificant spaces (3.4.1). The rule is identical to the caseExactSubstringsMatch rule except upper case characters are folded to lower case during string preparation as discussed in 3.2.

```
caseIgnoreSubstringsMatch MATCHING-RULE ::= {
    SYNTAX SubstringAssertion
    ID id-mr-caseIgnoreSubstringsMatch }
```

Both rules return TRUE if there is a partitioning of the prepared attribute value (into portions) such that:

- the specified substrings (initial, any, final) match different portions of the value in the order of the strings sequence.
- initial, if present, matches the first portion of the value;
- any, if present, matches some arbitrary portion of the value;
- final, if present, matches the last portion of the value.
- control is not used for the caseExactSubstringsMatch, caseIgnoreSubstringsMatch, telephoneNumberSubstringsMatch, or any other form of substring match for which only initial, any, or final elements are used in the matching algorithm; if a control element is encountered, it is ignored. The control element is only used for matching rules that explicitly specify its use in the matching algorithm. Such a matching rule may also redefine the semantics of the initial, any and final substrings.

  NOTE - The generalWordMatch matching rule is an example of such a matching rule.

There shall be at most one initial, and at most one final in the SubstringAssertion. If initial is present, it shall be the first element. If final is present, it shall be the last element. There shall be zero or more any elements.

For a component of substrings to match a portion of the attribute value, corresponding characters must be identical (including all combining characters in the combining character sequences).

4.4. Numeric String Match

The Numeric String Match rule compares for equality a presented numeric string with an attribute value of type NumericString.

```
numericStringMatch MATCHING-RULE ::= {
    SYNTAX NumericString
    ID id-mr-numericStringMatch }
```

The rule is identical to the caseIgnoreMatch rule (case is irrelevant

as characters are numeric) except that all space characters are removed during string preparation as detailed in Section 3.6.2.

4.5. Numeric String Ordering Match

The Numeric String Ordering Match rule compares the collation order of a presented string with an attribute value of type NumericString.

```
numericStringOrderingMatch MATCHING-RULE ::= {
    SYNTAX NumericString
    ID id-mr-numericStringOrderingMatch }
```

The rule is identical to the caseIgnoreOrderingMatch rule (case is irrelevant as characters are numeric) except that all space characters are removed during string preparation as detailed in Section 3.6.

4.6. Numeric String Substrings Match

The Numeric String Substrings Match rule determines whether a presented value is a substring of an attribute value of type NumericString.

```
numericStringSubstringsMatch MATCHING-RULE ::= {
    SYNTAX SubstringAssertion
    ID id-mr-numericStringSubstringsMatch }
```

The rule is identical to the caseIgnoreSubstringsMatch rule (case is irrelevant as characters are numeric) except that all space characters are removed during string preparation as detailed in Section 3.6.

4.7. Case Ignore List Match

The Case Ignore List Match rule compares for equality a presented sequence of strings with an attribute value which is a sequence of DirectoryStrings, without regard to the case (upper or lower) of the strings and insignificant spaces (3.6.1).

```
caseIgnoreListMatch MATCHING-RULE ::= {
    SYNTAX CaseIgnoreList
    ID id-mr-caseIgnoreListMatch }
CaseIgnoreList ::= SEQUENCE OF DirectoryString {ub-match}
```

The rule returns TRUE if and only if the number of strings in each is the same, and corresponding strings match. The latter matching is as for the caseIgnoreMatch matching rule.

4.8. Case Ignore List Substrings Match

The Case Ignore List Substring rule compares a presented substring with an attribute value which is a sequence of DirectoryStrings, but without regard for the case (upper or lower) of the strings and insignificant spaces (3.6.1).

```
caseIgnoreListSubstringsMatch MATCHING-RULE ::= {
    SYNTAX SubstringAssertion
    ID id-mr-caseIgnoreListSubstringsMatch }
```

A presented value matches a stored value if and only if the presented value matches the string formed by concatenating the strings of the stored value. This matching is done according to the caseIgnoreSubstringsMatch rule; however, none of the initial, any, or final values of the presented value are considered to match a substring of the concatenated string which spans more than one of the strings of the stored value.

4.9. Stored Prefix Match

The Stored Prefix Match rule determines whether an attribute value, whose syntax is DirectoryString, is a prefix (i.e. initial substring) of the presented value, without regard to the case (upper or lower) of the strings and insignficant spaces (3.6.1).

> NOTE - It can be used, for example, to compare values in the Directory which are telephone area codes with a value which is a purported telephone number.

```
storedPrefixMatch MATCHING-RULE ::= {
    SYNTAX DirectoryString {ub-match}
    ID id-mr-storedPrefixMatch }
```

The rule returns TRUE if the attribute value is an initial substring of the presented value with corresponding characters identical except with regard to case.

5. Other changes to X.520

This document makes the following changes to X.520:

The section 6.2.8 (Telephone Number Match) sentence:

> The rules for matching are identical to those for caseIgnoreMatch, except that all space and "-" characters are skipped during the comparison.

is replaced with:

The rules for matching are identical to those for caseIgnoreMatch, except that all hyphens and spaces are insignficant (3.6.3) and removed during the insignificant character removal step.

The section 6.2.9 (Telephone Number Substrings Match) sentence:

The rules for matching are identical to those for caseExactSubstringsMatch, except that all space and "-" characters are skipped during the comparison.

is replaced with:

The rules for matching are identical to those for caseExactSubstringsMatch, except that all hyphens and spaces are insignficant (3.6.3) and removed during the insignificant character removal step.

6. Security Considerations

See [RFC3454].

7. Acknowledgments

The approach used in this document is based upon design principles and algorithms described in "Preparation of Internationalized Strings ('stringprep')" [RFC3454] by Paul Hoffman and Marc Blanchet. Some additional guidance was drawn from Unicode Technical Standards, Technical Reports, and Notes.

Sections 3.3 and 4 of this document are derived from Section 6.1 of [X.520]. Additionally, some text was borrowed from [RFC3454].

This document is the product of IETF and ITU-T collaboration [IETF-ITU].

8. Editor's Address

Kurt Zeilenga
E-mail: <kurt@openldap.org>

9. References

9.1. Normative References

[RFC2119] S. Bradner, "Key words for use in RFCs to Indicate

Requirement Levels", BCP 14 (also RFC 2119), March 1997.

[RFC3454] P. Hoffman, M. Blanchet, "Preparation of Internationalized Strings ('stringprep')", RFC 3454, December 2002.

[X.501] International Telephone Union, "The Directory: The Models", X.501, 2000.

[X.520] International Telephone Union, "The Directory: Selected Attribute Types", X.520, 2000.

[ISO10646] Universal Multiple-Octet Coded Character Set (UCS) - Architecture and Basic Multilingual Plane, ISO/IEC 10646-1 : 1993.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 3.2.0" is defined by "The Unicode Standard, Version 3.0" (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5), as amended by the "Unicode Standard Annex #27: Unicode 3.1" (http://www.unicode.org/reports/tr27/) and by the "Unicode Standard Annex #28: Unicode 3.2" (http://www.unicode.org/reports/tr28/).

[UAX15] M. Davis, M. Duerst, "Unicode Standard Annex #15: Unicode Normalization Forms, Version 3.2.0". <http://www.unicode.org/unicode/reports/tr15/tr15-22.html>, March 2002.

[T61-UCS] TBD

9.2. Informative References

[X.500] International Telephone Union, "The Directory: Overview of Concepts, Models and Service", X.500, 2000.

[IETF-ITU] G. Fishman, S. Bradner, "Internet Engineering Task Force and International Telecommunication Union - Telecommunications Standardization Sector Collaboration Guidelines", TSAG A-Series Supplement 3, November 2001 (also RFC 3356, published August 2002).

[GLOSSARY] The Unicode Consortium, "Unicode Glossary", <http://www.unicode.org/glossary/>.

[UTR17] K. Whistler, M. Davis, "Unicode Technical Report #17, Character Encoding Model", UTR17, <http://www.unicode.org/unicode/reports/tr17/>, August

2000.